Санкт-Петербургский государственный университет

Кафедра технологии программирования

Нефедова Екатерина Алексеевна

Выпускная квалификационная работа бакалавра

«Анализ тональности текстов с использованием нейросетевых моделей»

Направление 010400

«Прикладная математика и информатика»

Научный руководитель,

ст. преподаватель

Мишенин А.Н.

Санкт-Петербург

2016

Содержание

Введение

Человек – существо феноменальное, но вся его уникальность кроется в мозге. Этому серому веществу подвластно абсолютно все: от простейших математических примеров до неизведанных границ. Но, к сожалению, люди не вечны, также они не могут постоянно работать без отдыха; уставший человек теряет внимательность и может совершить ошибки даже в самых простых задачах.

На дворе век информационных технологий, компьютеры могут многое, в чем-то даже превосходят человеческие возможности. Но научить машину думать, как человек, – мы к этому только стремимся!

При рождении ребенка мозг имеет совершенную структуру, благодаря которой создаются собственные правила под воздействием опыта. Накопление опыта идет постоянно, наиболее сильные изменения происходят в первые годы жизни ребенка. Однако развитие продолжается до последних дней жизни человека. Это обусловлено пластичностью мозга – способностью настройки нервной системы в соответствии с окружающими условиями. Именно эта функция играет важную роль в работе нейронов в качестве единиц обработки информации в мозге человека.

В середине прошлого столетия началось распространение и изучение искусственных нейронных сетей, начало которым было дано в статье 1943г. *McCalloch W.S., Pitts W.* «*A logical calculus of the ideas immanent in nervous activity*».

Основной идеей создания искусственных нейронных сетей стала аналогия с устройством нейронной сети в человеческом мозге, то есть вся работа осуществляется при помощи нейронов. В общем случае нейронная сеть представляет собой модель человеческого мозга, решающую поставленную задачу.

Нейронная сеть – это громадный распределенный параллельный процессор, состоящий из элементарных единиц обработки информации, накапливающих экспериментальные знания и предоставляющих их для последующей обработки. Нейронная сеть сходна с мозгом с двух точек зрения:

- знания поступают в нейронную сеть из окружающей среды и используются в процессе обучения;
- для накопления знаний применяются связи между нейронами, называемыми синаптическими весами.

Основой данной работы является исследование методов глубинного изучения (*deep learning*) с использованием нейросетевых моделей для решения задач обработки текстов. Для того чтобы проводить исследования, необходимо сначала перевести естественный язык в понятный для компьютера формат, в данном случае – числовой. Для представления слов и документов в векторном виде будем использовать различные модели, такие как «*мешок слов*» (*Bag of Words*), *Word2Vec*, *Doc2Vec*.

В качестве примера обработки текста была выбрана задача определения тональности рецензий пользователей сервиса *Kinopoisk*. Определение тональности текстов является весьма актуальной задачей. Ежедневно тысячи пользователей штудируют Интернет в поисках мнений о том или ином товаре, услугах организаций и прочее. Отзывы помогают определиться с выбором не только людям, но и компаниям, что также полезно. При помощи отзывов организация может судить о качестве своей работы. Естественно, не стоит забывать о более глобальных задачах, например, исследование политических настроений в преддверии выборов или оценка существующей власти.

Задача подразумевает обучение классификаторов на имеющемся множестве размеченных данных, которые будут разделены на два

подмножества: обучающее и тестовое. Каждое подмножество представлено в виде текста рецензии на русском языке и оценки, несущей позитивный или негативный мотив.

Постановка задачи

Цель работы – исследование методов глубинного изучения (deep learning) и их применение к задачам обработки текстов. Для достижения поставленной цели необходимо решить следующие задачи:

1. обзор современных методов машинного обучения, основанных на применении многослойных нейронных сетей, и их применение к задачам обработки текстов;
2. анализ преимуществ и недостатков существующей реализации алгоритма *Word2Vec*;
3. сравнение изученных моделей, применительно к задаче определения тональности рецензий пользователей сервиса *Kinopoisk*.

Обзор литературы

Для подготовки к написанию данной работы было изучено немало материалов. Основой основ являются нейросетевые модели. В книге «*Neural Networks a Comprehensive Foundation*», *S. Haykin* [5] очень подробно рассказывается, что такое нейронные сети, каких типов они бывают, о способах обучения и многом другом. В работе использовалась только небольшая часть из введения для определения основных понятий для нейронных сетей. Нейронные сети обладают проблемой исчезающего градиента, но есть сети, где этот недостаток устранен. В статье «*Understanding LSTM Networks*» блога *Christopher Olah* [1] описывается структура и принцип работы такой рекуррентной нейронной сети как *Long Shot Term Memory networks* (*LSTM*).

Также существуют модели, основанные на нейронных сетях. В статье «*Efficient Estimation of Word Representations in Vector Space*», *T. Mikolov, K. Chen, G. Corrado, J. Dean* [2] были представлены два нестандартные архитектуры для вычисления непрерывных векторных представлений слов на очень большом объеме данных: *Continuous Bag-of-Words* (*CBOW*), *Continuous Skip-gram*. В статье было произведено сравнение качества получаемых представлений с результатами моделей, ранее считавшиеся самыми эффективными. Результаты экспериментов показали, что новая модель для получения представления слов показывает лучшие результаты для синтаксического и семантического сходства слов, а также занимает гораздо меньше времени для обучения. В статье «*Distributed Representations of Word and Phrases and their Compositionaly*», *T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean* [3] более подробно описывается архитектура *Skip-gram*, а также представлены некоторые дополнения для улучшения качества векторов и увеличения скорости обучения. Были проведены эксперименты, где не только словам сопоставлялись векторы, но и фразам. Фразы были именно такие, что при разделении на слова, сразу терялся смысл. Было

показано, что обучение хороших векторных представлений фраз также возможно.

В [2] и [3] речь идет именно о векторных представлениях слов, следующая работа была навеяна предыдущими. Идея заключается в том, что не только словам сопоставляются векторы, но и документам, которым они принадлежат. В статье «*Distributed Representations of Sentences and Documents*», *T. Mikolov, Q. Le* [4] описываются архитектуры для получения векторных представлений для документов переменной длины: *Distributed Memory* (*DM*), *Distributed Bag-of-Words* (*DBOW*). Полученные векторы также имеют семантическую близость, но при этом решена проблема представления документа в числовом виде, которая возникает при обращении к алгоритмам машинного обучения.

Для сравнения различных способов анализа тональности текстов нельзя было не упомянуть про классический подход, основанный на словаре. Для русского языка в свободном доступе имеется очень мало размеченных данных, в частности корпусов, где каждому термину сопоставлена тональность. Из веб-ресурса http://linis-crowd.org [7] был взят размеченный словарь, который включает в себя порядка 7000 слов.

В данной работе в качестве данных для обработки были использованы рецензии к фильмам, взятые с сервиса http://www.kinopoisk.ru [6].

Глава 1. Описание предметной области

1.1 Анализ тональности текста

Мнения занимают основное место почти во всех областях человеческой деятельности. Наши убеждения и представления о реальности и выбор, который мы делаем, в значительной мере зависит от того, как другие видят и оценивают мир. По этой причине, когда нам нужно принять решение, мы часто ищем чужие мнения. Это справедливо не только для людей, но и для организаций. Мнения и связанные с ними понятия, такие как чувства, оценки, отношения и эмоции являются предметом изучения анализа настроений (*sentiment analysis*). Зарождение и быстрое развитие этой области связано с интересами людей в Интернете, как правило, спрос всегда порождает предложение. В качестве примера можно привести различные отзывы, форумы, обсуждения, блоги, социальные сети. Впервые в человеческой истории мы имеем огромный объем мнений, записанных в цифровом формате. С начала XXI века сфера анализа тональности данных стала одной из наиболее активно развивающихся и исследуемых направлений в области обработки естественных языков.

Анализ тональности текста – обработка естественного языка, классифицирующая тексты по эмоциональной окраске. Такой анализ можно рассматривать как метод количественного описания качественных данных, с присвоением оценок настроения. Целью является нахождение мнений в тексте и определение их свойств. Например, необходимо выявить автора текста – определить субъект, затем то, о чем ведется речь – объект разговора, и, наконец, отношение первого ко второму – определение тональности. Отзывы в Интернете упрощают жизнь, так как будь то текст из социальной сети или отзыв о товаре, автор мнения чаще всего известен, как и сам объект суждения. Остается самое интересное – отношение автора к объекту. Для этого нужно понять смысл текста, что является весьма непростой задачей.

Задача определения эмоциональной окраски текста является задачей классификации, она может быть бинарной (негативный, позитивный), тернарной (негативный, нейтральный, позитивный) или n-арной (например, для $n = 5$: сильно негативный, умеренной негативный, нейтральный, умеренно позитивный, сильно позитивный).

Формализация задачи анализа тональности текста выглядит следующим образом. Пусть X – множество всех документов, Y – множество меток размера N, $\varphi: X \to Y$ – целевая функция определения тональности документов, значения которой известны только для обучающего подмножества $X_{train} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$. Необходимо найти алгоритм $a: X \to Y$, который способен классифицировать произвольный документ $x \epsilon X$, расставить метки.

При решении задачи возникает ряд трудностей, он может быть связан с порядком слов в предложении, омонимичностью слов, орфографическими и синтаксическими ошибками, все это может в корне менять смысл, а следовательно, и эмоциональную окраску.

В общем случае тональность текста весьма субъективна, но ее анализ находит много полезных применений. Например, компании могут понять реакцию пользователей на продукт, а также выявить негативные комментарии.

1.2 Компьютер меня не понимает

Компьютер считает быстрее человека, но гораздо хуже понимает естественный язык. Например, существуют два слова, обозначающие одно и то же. Человек, опираясь на свой жизненный опыт, на свои знания, может понять, что эти слова имеют одинаковую смысловую нагрузку. Компьютер же «думает» совсем иначе, сразу встает вопрос, как научить понимать компьютер, что значение этих слов одинаково. На помощь приходит компьютерная лингвистика, при помощи которой можно произвести фонетический, морфологический и синтаксический анализ слов. В итоге анализа какого-нибудь слова получаются всевозможные локальные репрезентации в виде фонетического разбора на звуки, принадлежность слова к части речи, роду, числу, падежу, а также его роль в предложении. Получено очень много важных свойств о слове, но ничего о его значении.

Для того чтобы подобраться к смысловому значению слова, необходимо произвести семантический анализ. Необходимо смоделировать, у каких слов схожее значение. Если слова обозначают одно и то же, то и их репрезентации должны быть похожи.

Существует два фундаментальных подхода к моделированию семантики:

1. Подход, построенный на знаниях. Также этот подход можно назвать подходом «сверху вниз». Это трудоемкий способ, требующий огромных человеческих ресурсов. В данном случае тысячи экспертов должны построить онтологическую модель, описав какие слова являются синонимами, антонимами, смысловыми подчастями других слов и т.д.
2. Дистрибутивный подход или подход «снизу вверх». Здесь извлекается значение из употребления слов в тексте.

Рассмотрим более подробно второй подход. Человеческий мозг хранит информацию о всевозможных контекстах, в которых мы видели или слышали какое-то слово, таким образом мы узнаем его смысловую нагрузку. Отсюда можно сделать вывод, что слова, встречающиеся в схожих контекстах, имеют схожее значение. Аналогично будем «объяснять» компьютеру, основываясь на различных текстах (в данном случае, чем их будет больше, тем лучше) и модели, которая описывает совместную встречаемость слов.

Традиционные счетные модели

Алгоритм построения модели:

1. Составляем словарь из всех слов, встречающихся в тексте, предварительно исключив из него все знаки препинания и «стоп-слова».
2. Для каждого слова записываем его соседей, то есть слова, стоящие рядом.
3. Каждое слово представляем в виде последовательности чисел, а точнее в виде вектора. Размерность этого вектора совпадает с количеством всех слов в корпусе. Каждая компонента вектора соответствует определенному слову, значение которой – совместная встречаемость этих слов.

Можно использовать не абсолютную частоту совместной встречаемости, а взвешенную. Например, используя коэффициент Дайса:

$$Dice(A,B) = \frac{2 * c(A,B)}{c(A) + c(B)},$$

где $c(A)$ – абсолютная частота слова A, $c(B)$ – абсолютная частота слова B, $c(A,B)$ – частота совместной встречаемости.

В итоге получаем нормированный вес. Это значение позволяет учесть, что слова могут стоять рядом не только потому, что являются каким-то устойчивым выражением, а просто часто встречаются в словаре.

Так как каждое слово представлено в виде вектора, то их можно изобразить графически, следовательно мы получим в общем случае N-мерное пространство, если N – количество слов в корпусе. Можно заметить, что похожие слова, располагаются рядом, то есть имеют схожие вектора. Для их нахождения будем вычислять семантическую близость с использованием косинусной близости векторов.

Функция косинусной близости представляет собой скалярное произведение нормированных векторов:

$$\cos(A, B) = \frac{\vec{V}(A) * \vec{V}(B)}{\left\|\vec{V}(A)\right\| * \left\|\vec{V}(B)\right\|},$$

где $\vec{V}(A) * \vec{V}(B)$ – скалярное произведение векторов, $\left\|\vec{V}(A)\right\| * \left\|\vec{V}(B)\right\|$ – произведение евклидовых норм векторов.

Из этого равенства делаем вывод, что чем больше значение функции косинусной близости двух векторов, тем больше их схожесть, и наоборот.

Словарный подход

Еще одним способом определения тональности текста является метод, основанный на словаре. Заранее имеется словарь размеченных данных, в котором каждому слову соответствует определенное настроение. При этом каждое слово в документе вносит свой определенный вклад в смысл и эмоциональную окраску текста, имеет свой вес. Для назначения каждому слову веса существуют различные виды статистических мер, например, $tf - idf$:

$$tf(t,d) = \frac{n_i}{\sum_k n_k},$$

$$idf(t,D) = \log\frac{|D|}{|t \subset d|},$$

где n_i – число вхождений слова t в документ d, $\sum_k n_k$ – общее количество всех слов в документе, $|D|$ – количество документов в корпусе, $|t \subset d|$ – количество документов, в которых встречается слово t.

Мера $tf - idf$ имеет следующий вид:

$$tf - idf(t,d,D) = tf(t,d) * idf(t,D).$$

В итоге тональность текста считается следующим образом: берется сумма по всем словам текста от произведения тональности слова на его вес.

Недостатки модели:

- требуются заранее размеченные словари большого объема;
- неуниверсальность модели, то есть в разном контексте слова могут иметь разную тональность.

Глава 2. Модели для построения векторного представления

2.1 «Мешок слов»

Одним из распространенных подходов является «*мешок слов*» (*Bag-of-Words*). «*Мешок слов*» – это модель, которая обучается на словаре, составленном из слов всех документов.

Алгоритм построения модели:

1. Составляем словарь из всех слов, встречающихся в тексте, предварительно исключив все знаки препинания, числа и «стоп-слова».
2. Для каждого документа определяем вектор, каждая компонента которого соответствует термину из словаря, а ее значение определяется числом, сколько раз это слово встретилось в тексте. Размерность вектора соответствует мощности словаря.

Например, есть 2 документа:

1. «Саша любит смотреть фильмы в кинотеатре. Маша любит смотреть фильмы дома»;
2. «Саша не любит смотреть дома футбольные матчи».

Словарь будет выглядеть следующим образом: {«Саша», «любит», «смотреть», «фильмы», «в», «кинотеатре», «Маша», «дома», «не», «футбольные», «матчи»}. Теперь запишем векторные представления документов согласно словарю:

1. [1, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0]
2. [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]

Недостатки модели:

- очень большой размер векторов;
- замедление операции сравнения векторов из-за размерности векторов;
- можно применять различные методы снижения размерности, но это приведет к потере качества.

2.2 Word2Vec

В данной модели для получения хороших векторов используется машинное обучение. Одним из популярных методов является построение искусственных нейронных сетей. Изначально задается размерность векторов, которые заполняются случайными величинами. Во время обучения значения компонент векторов будут меняться, причем вектор каждого слова будет максимально схож с векторами типичных соседей и максимально отличаться от векторов слов, которые соседями данному слову не являются. Сами компоненты векторов никак не связаны с конкретными словами из словаря. Но и здесь не все так гладко, при обучении нейронных сетей требуется очень много времени и огромные вычислительные затраты.

Но в 2013 году произошла своего рода революция, *Tomas Mikolov* вместе с соавторами опубликовал статью «*Efficient Estimation of Word Representations in Vector Space*» [2], а позже выложил код утилиты *Word2Vec*, которая позволяет тренировать нейронные языковые модели на больших словарях. Миколов модифицировал существовавшие до этого алгоритмы, поэтому *Word2Vec* обучается на порядок быстрее, чем нейронные языковые модели до него.

Алгоритм построения модели:

1. Составляется словарь терминов, встретившихся во всех документах. Его размер практически не ограничивается, исключаются только слова, имеющие наименьшую встречаемость.
2. Каждому термину в словаре сопоставляется частота встречаемости во всех документах
3. Для кодирования словаря строится дерево Хаффмана.
4. Производится субдискретизация частых слов (параметр задается при создании модели).

5. Для этих слов применяется один из алгоритмов *CBOW* (*Continuous Bag-of-Words*) или *Skip-gram*.
6. Применяется нейронная сеть прямого распространения с функцией активации *иерархический softmax* или *негативное семплирование* (*negative sampling*).

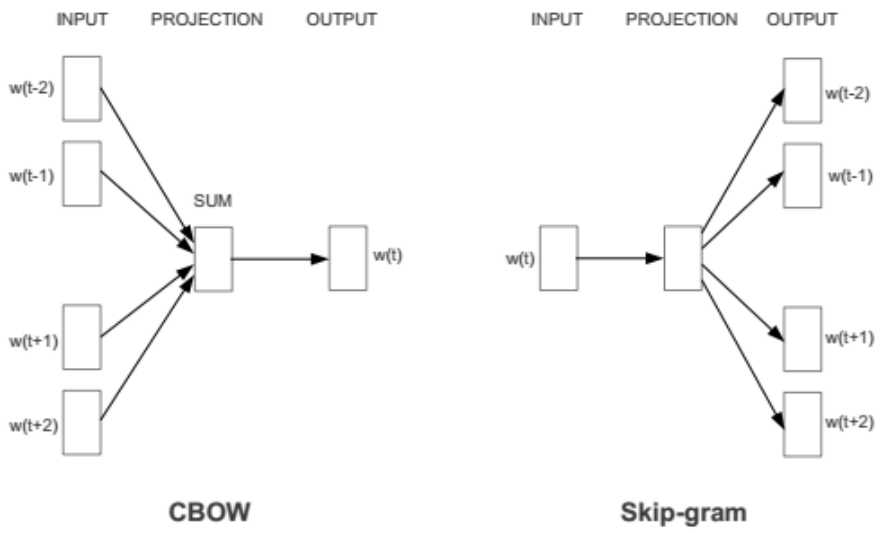

Рис. 1. Архитектуры модели Word2Vec: CBOW и Skip-gram.

На Рис. 1 представлены архитектуры модели *Word2Vec*. Архитектура *CBOW* аналогична нейронной сети прямого распространения, где нелинейный скрытый слой удаляют, а проекция слоя является общей для всех слов, таким образом, все слова находятся в одинаковом положении. Очень похоже на модель «*мешок слов*» (*Bag-of-Words*): порядок слов никак не влияет на проекции. Задача архитектуры при обучении модели – предсказать слово по имеющемуся контексту. *CBOW* в отличие от *BOW* использует непрерывное распределенное представление контекста, поэтому и имеет такое название.

Архитектура *Skip-gram* похожа на *CBOW*, но вместо того, чтобы предсказывать текущее слово исходя из окружения, она пытается для текущего слова предсказать контекст – слова, стоящие в пределах определенного диапазона (до и после данного слова). Увеличение

контекстного окна улучшает качество получаемых векторов, но заметно увеличивает вычислительную сложность. Архитектура *Skip-gram* также представлена на Рис. 1.

Использовать данную утилиту может любой желающий, не имеющий суперкомпьютера, что, безусловно, является огромным плюсом, по сравнению с нейронными сетями. Достаточно лишь взять достаточно большой обучающий корпус, использовать *Word2Vec* и получить хорошие вектора, чтобы в дальнейшем, например, поможет находить синонимы. Для сравнения двух векторов здесь также используется функция косинусной близости.

Модели, натренированные на очень большом количестве текстов, показывают удивительные результаты: алгебраические операции на векторах отображают семантические операции. Получается своего рода семантический калькулятор.

Если мы из вектора слова «Париж» вычтем вектор слова «Франция», а затем добавим вектор слова «Германия», то есть:

$$\text{Париж} - \text{Франция} + \text{Германия} \cong \text{Берлин},$$

то в результате алгебраических операций мы получим вектор, наиболее схожий с вектором слова «Берлин».

Применение *Word2Vec*:

- вычисление семантической близости;
- машинный перевод;
- расширение поисковых запросов;
- классификация текстов на заранее заданные категории;
- кластеризация текстов на заранее неизвестные категории;
- определение тональности высказывания (например, необходимо определить положительный или отрицательный отзыв).

Распределенные векторы слов являются мощным инструментом и используются для многих приложений, в особенности для предсказаний слов и перевода. Здесь мы постараемся применить их для анализа тональности [2, 3].

2.3 Doc2Vec

Многие методы машинного обучения требуют на вход данные, представленные в виде вектора признаков фиксированной длины. Когда дело доходит до текстов одним из самых распространенных способов представления векторов фиксированной длины является «*мешок слов*». Однако эта модель имеет много недостатков. Порядок слов теряется, разные предложения, имеющие одни и те же слова, могут иметь одинаковые представления. Алгоритм *Doc2Vec* решает эту проблему.

Алгоритм *Doc2Vec* (первоначальное название *Paragraph Vector*), алгоритм обучения без учителя, учится получать распределенные векторы для частей текстов. Тексты могут быть переменной длины: от предложения до большого документа. Для простоты все входные тексты в данном разделе будут именоваться документами.

В данной модели векторные представления документов обучаются предсказывать слова в документе, точнее берется вектор документа и объединяется с несколькими векторами слов из него, и модель пытается предсказать следующее слово с учетом контекста. Векторы слов и документов обучаются с использованием метода стохастического градиентного спуска и метода обратного распространения ошибки. Векторы документов являются уникальными, а векторы одинаковых слов в разных документах совпадают.

Существует 2 архитектуры для построения векторных представлений документов:

1. *Distributed Memory* (*DM*, *D2V-DM*);
2. *Distributed Bag-of-Words* (*DBOW*, *D2V-DBOW*).

D2V-DM

В данной конструкции каждый документ представлен уникальным вектором в виде столбца в матрице D, и каждый термин представлен уникальным вектором в виде столбца в матрице W. Вектор документ и векторы слов в нем объединяются или усредняются для предсказания следующего слова из контекста. На Рис. 2 представлена графическая интерпретация данной архитектуры.

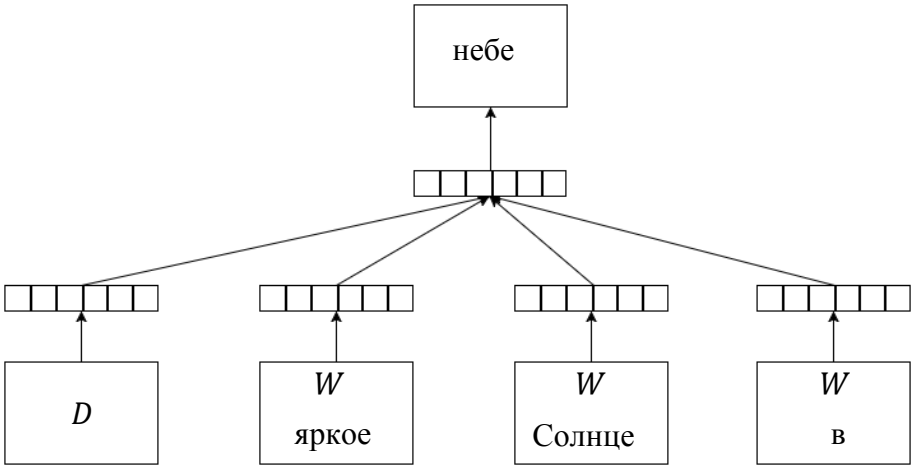

Рис. 2. Архитектура модели Doc2Vec - DM (distributed memory).

В этой архитектуре можно рассматривать токены как отдельные слова. Они действуют как память, которая помнит, что отсутствует в текущем контексте или теме документа. По этой причине модель называется *Distributed Memory*.

D2V-DBOW

Описанный выше метод рассматривает объединение вектора документа с векторами слов, входящих в него, для предсказания следующего слова в текстовом окне. Другой способ заключается в игнорировании слов из контекста на входе, но при этом модель должна предсказывать случайно отобранные слова для документа на выходе. Это означает, что на каждой итерации стохастического градиентного спуска просматривается текстовое окно, затем просматривается случайное слово в текстовом окне и

формируется задача классификации с учетом вектора документа. Эта версия называется *Distributed Bag-of-Words*. На Рис. 3 представлена графическая интерпретация данной архитектуры [4].

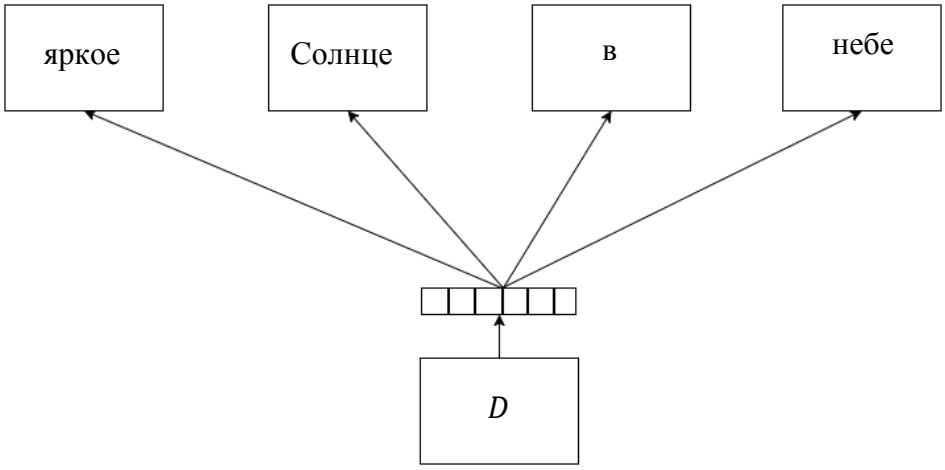

Рис. 3. Архитектура модели Doc2Vec - DBOW (distributed bag of words).

Глава 3. Классификация

3.1 Рекуррентные нейронные сети

Основные понятия. Модель нейрона

Нейрон – единица обработки информации в нейронной сети.

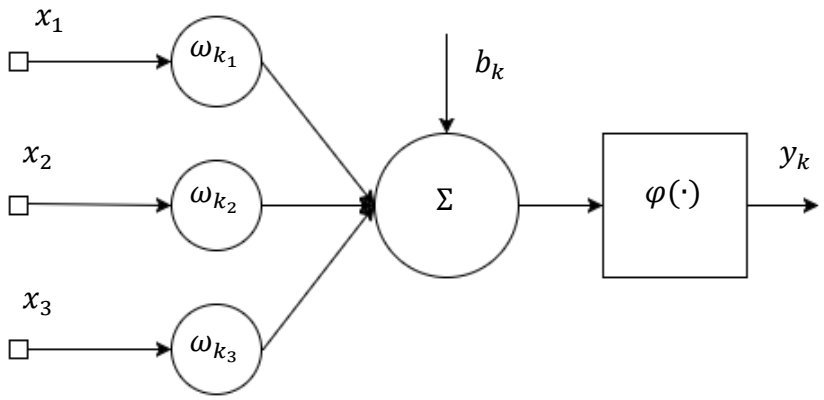

Рис. 4. Модель нейрона.

Основные элементы нейрона:

1. набор синапсов, каждый из которых характеризуется своим весом. Данные x_j, подающиеся на вход j, для нейрона k, умножаются на вес ω_{k_j};
2. сумматор образует линейную комбинацию, путем складывания взвешенных входных данных относительно синапсов нейрона;
3. функция активации или функция сжатия ограничивает амплитуду выходного сигнала;
4. пороговый элемент – величина, отражающая увеличение или уменьшение входного сигнала, подаваемого на функцию активации.

На математическом языке обработку входных данных нейроном можно описать следующим образом:

$$u_k = \sum_{j=1}^{m} \omega_{k_j} x_j,$$

$$y_k = \varphi(u_k + b_k),$$

где $x_1, x_2, \dots, x_m$ – входные данные, $\omega_{k_1}, \omega_{k_2}, \dots, \omega_{k_m}$ – синаптические веса нейрона k, u_k – линейная комбинация входных воздействий, b_k – порог, $\varphi(\cdot)$ – функция активации, y_k – выходной сигнал нейрона k. В качестве аргумента функции активации используется постсинаптический сигнал v_k.

$$v_k = u_k + b_k$$

Типы функций активации:

1. Функция Хэвисайда (функция единичного скачка, пороговая функция)

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$

2. Кусочно-линейная функция

$$\varphi(v) = \begin{cases} 1, v \geq {}^1\!/_2 \\ |v|, -{}^1\!/_2 < v < {}^1\!/_2 \\ 0, v \leq -{}^1\!/_2 \end{cases}$$

3. Сигмоидальная функция

Пример – логистическая функция

$\varphi(v) = \frac{1}{1+e^{(-av)}}$, a – параметр наклона

Одно из важных свойств нейронных сетей – способность обучаться на основе входных данных для повышения своей производительности. Обучение – это процесс, в котором свободные параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки этих параметров.

Схема нейронной сети тесно связана с алгоритмами обучения. В общем случае можно выделить три класса:

1. Однослойные сети прямого распространения.
2. Многослойные сети прямого распространения.
3. Рекуррентные сети.

Сетью прямого распространения или ациклической сетью называется такая нейронная сеть, в которой информация поступает на входной слой и передается на выходной, но не наоборот.

Однослойные сети прямого распространения

В данном случае единственным слоем нейронной сети является слой вычислительных нейронов, входной и выходной слои совпадают. На Рис. 5 представлена архитектура однослойной нейронной сети прямого распространения.

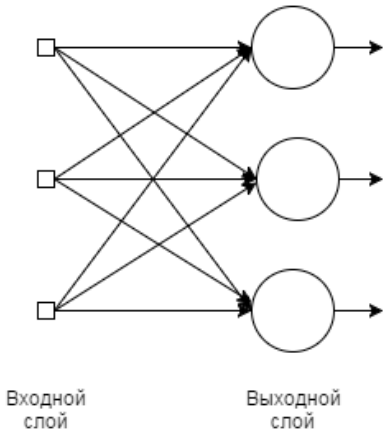

Рис. 5. Архитектура однослойной нейронной сети прямого распространения.

Многослойные сети прямого распространения

Данный класс сетей определяется наличием одного или нескольких скрытых слоев, узлы которых называются скрытыми нейронами. Они несут функцию посредников между внешним входным сигналом и выходом. Обычно нейроны каждого из слоев сети используют в качестве входных данных только выходные данные предыдущего слоя. Если все узлы каждого

конкретного слоя соединены со всеми узлами смежных слоев, то сеть называется полносвязной, в противном случае – неполносвязной. На Рис. 6 изображена архитектура многослойной сети прямого распространения 5-3-2 (название дается по количеству нейронов на каждом слое) [5].

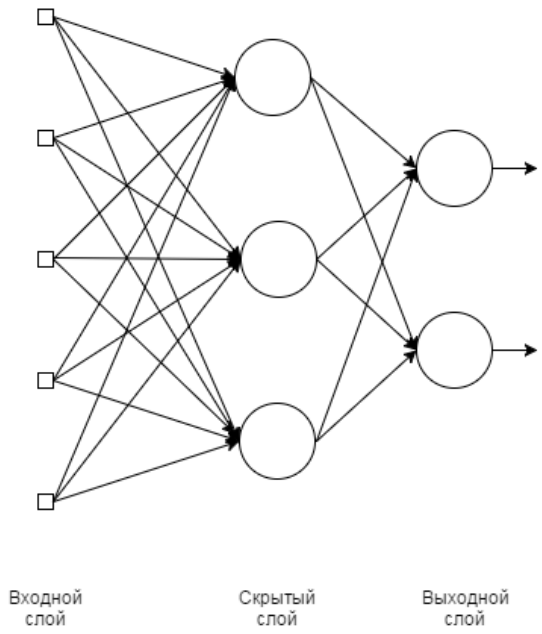

Рис. 6. Архитектура многослойной нейронной сети прямого распространения 5-3-2.

Рекуррентные нейронные сети

Люди не начинают думать с нуля каждую секунду. Когда вы читаете текст, вы понимаете каждое слово, основываясь на понимании предыдущих слов. Вы не выбросите все и не начнете заново думать с нуля, ваши мысли сохраняются.

Традиционные нейронные сети не могут сделать это, что является серьезным недостатком. Представьте, что хотите классифицировать, какое событие происходит в каждой точке фильма. Пока неясно, как традиционная нейронная сеть может использовать свои рассуждения о предыдущих событиях в фильме, чтобы сообщить более поздние. Рекуррентные нейронные сети решают эту проблему. Они представляют собой сети с циклом, что позволяет информации сохраняться.

На схеме (Рис. 7) изображена часть нейронной сети. Цикл позволяет информации передаваться от одного шага сети к другому. Нейроны рекуррентной сети выглядят как обычные, но при этом у них есть дополнительная циклическая стрелка. По сути, рекуррентную нейронную сеть можно рассматривать как несколько копий одной и той же сети, каждая из которых передает сообщение следующей. Графически это можно показать следующим образом.

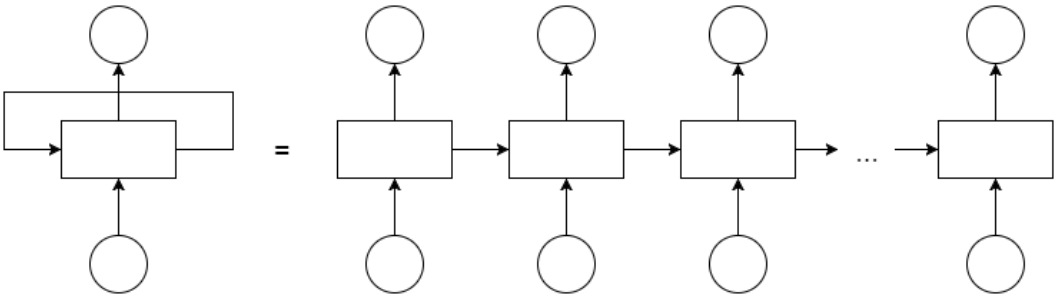

Рис. 7. Часть рекуррентной нейронной сети.

На вход таким сетям обычно поступают последовательная информация, например, тексты, аудио- и видеосигналы. При помощи подобной архитектуры можно решать такие задачи, как предсказание последнего слова в предложении. Иногда нужна только последняя информация для выполнения данной задачи. Например, в предложении «облака в *небе*» попытаемся предсказать последнее слово. В данном случае не нужен никакой дополнительный контекст, это довольно очевидно, что искомое слово будет «небо». Но есть и другие ситуации, например, решая ту же задачу, но уже в тексте: «Я вырос во Франции… Я свободно говорю по-французски». Последняя информация говорит о том, что следующее слово, вероятно, является названием языка. Но для конкретики, какой язык, необходим контекст из прошлого, в данном случае – это Франция. Вполне возможно, что зазор (расстояние) между соответствующей информацией и точкой, где это необходимо, может быть очень большим. К сожалению, с увеличением разрыва рекуррентная нейронная сеть оказывается не в состоянии научиться соединять информацию. Следует отметить, что в пределах одного

предложения такая сеть может работать весьма неплохо для предсказания слов, но для больших текстов это становится невозможным. Теоретически рекуррентная нейронная сеть способна обрабатывать такие «долгосрочные зависимости». Человек мог бы тщательно подобрать параметры для сети, чтобы решить эту проблему. Но на практике сеть не способна обучиться этому. Данная проблема была рассмотрена *Josef Hochreiter* в работе «*Untersuchungen zu dynamischen neuronalen Netzen*» и *Yoshua Bengio* в работе «*Learning Long-Term Dependencies with Gradient Descent is Difficult*» и была названа проблемой исчезающего градиента. Это проблема не только рекуррентных нейронных сетей, это фундаментальная проблема всех нейронных сетей. Как следствие развитие и изучение нейронных сетей в конце XX века приостановилось, и они уступили лидерство машинам опорных векторов и алгоритмам бустинга.

Но не все так плохо. В 1997 году *Hochreiter & Schmidhuber* в работе «*LONG SHORT-TERM MEMORY*» была введена в пользование сеть под названием *Long Short Term Memory networks* (*LSTM*). *LSTM* – особый вид рекуррентных нейронных сетей, которые способны к обучению долгосрочных зависимостей. Они работают чрезвычайно хорошо на различных типах задач и сейчас широко применяются.

LSTM-RNN предназначена для решения проблемы долгосрочной зависимости. Сохранение информации в течение длительного времени – это практически их поведение по умолчанию.

Все рекуррентные нейронные сети имеют вид цепочки повторяющихся модулей нейронной сети. В стандартных рекуррентных нейронных сетях это повторение модуля будет иметь очень простую структуру: сложение двух векторов (сигнала и памяти), вычисление функции активации от суммы. Получается обычная сеть с одним скрытым слоем. Схема изображена на Рис. 8.

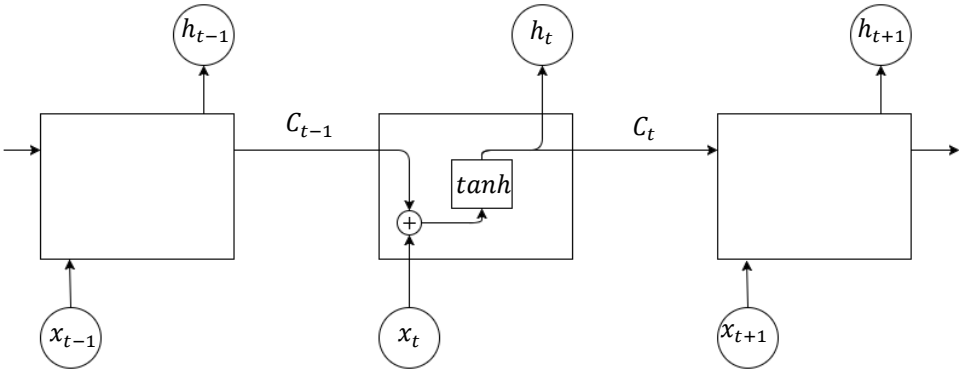

Рис. 8. Структура работы рекуррентной нейронной сети.

LSTM имеет по структуре такую же цепь, но «внутренность» повторяющегося модуля выглядит иначе. Отличие сети состоит в том, как обрабатывается ячейка памяти. Вместо того чтобы иметь единственный слой нейронной сети, здесь их четыре, при этом они взаимодействуют особым образом. Схема изображена на Рис. 9.

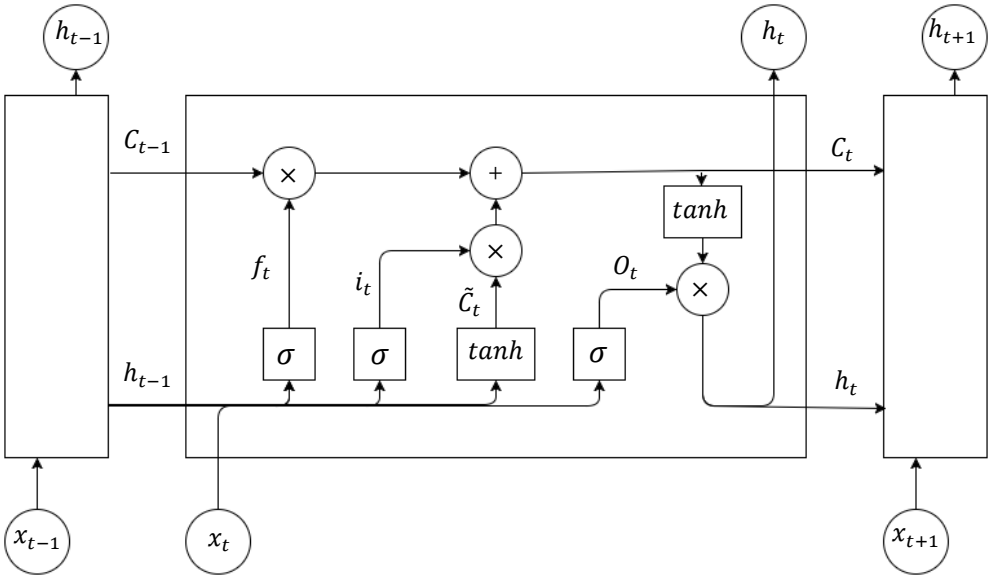

Рис. 9. Структура работы рекуррентной нейронной сети LSTM.

Разберем каждый из слоев. На первом слое нужно решить, забывать предыдущую информацию или нет. Это решение принимает сигмоидальный слой. Ему на вход поступают значения h_{t-1}, x_t, на выходе имеется число от 0 до 1, в градации 0 – удалить навсегда, 1 – запомнить навсегда.

$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right)$$

На втором слое принимается решение, насколько важна пришедшая новая информация. Сигмоидальный слой решает, какие значения будут обновлены. Слой с гиперболическим тангенсом создает вектор новых значений $\tilde{C}_t$, которые могут быть добавлены.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C)$$

На третьем слое происходит обновление старого состояния ячейки C_{t-1} в новое состояние ячейки C_t. Для этого необходимо умножить значение старого состояния на f_t и добавить произведение значений, полученных на предыдущем слое.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

На последнем слое производится решение, какое значение пойдет на выход. Сначала сигнал проходит через сигмоидальный слой, который решает, какую часть нужно оставить для дальнейшего решения. Затем гиперболический тангенс «размазывает» вектор памяти на отрезок от -1 до 1. Полученные векторы перемножаются.

$$O_t = \sigma(W_O[h_{t-1}, x_t] + b_O)$$

$$h_t = O_t \cdot \tanh(C_t)$$

Полученные значения C_t и h_t передаются дальше по цепочке. Существует много вариантов для выбора функций активации, но суть всегда остается одна: сначала забывают часть памяти, затем запоминают часть нового сигнала [1].

3.2 SVM (support vector machine) – метод опорных векторов

Одним из популярных бинарных классификаторов является метод опорных векторов. Основной идеей метода является перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости в новом пространстве. Является контролируемым методом обучения. На Рис. 10 представлена графическая интерпретация метода опорных векторов. На рисунке изображено несколько разделяющих гиперплоскостей, но только одна достигает оптимального разделения.

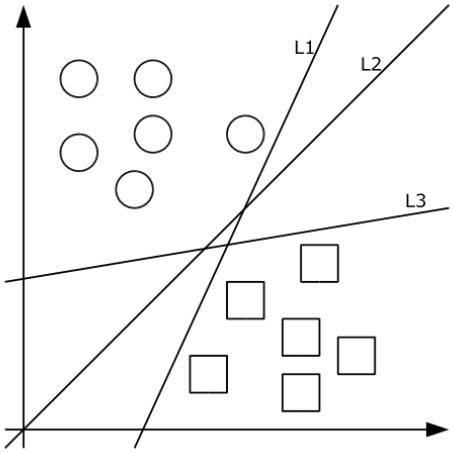

Рис. 10. Графическая интерпретация метода опорных векторов.

Стоит отметить, что при использовании этого метода необходимо масштабировать данные. Масштабирование – это часть процесса стандартизации, нормировка данных, набор приводим к нормальному гауссовому распределению (математическое ожидание $\mu = 1$, среднеквадратическое отклонение $\sigma = 1$). Новые значения рассчитываются по следующей формуле:

$$x_{new} = \frac{x - \mu}{\sigma},$$

где μ – математическое ожидание, σ – среднеквадратическое отклонение.

Также может использоваться нормализация:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}},$$

где x_{min} и x_{max} – минимальное и максимальное значение данных соответственно.

Преимущества *SVM*:

- эффективность в пространствах высокой размерности;
- использование различных ядер для функции принятия решения;
- использование подмножества точек в функции решения (так называемые опорные векторы);
- эффективность при условии, что число измерений векторов превосходит число образцов.

Недостатки:

- вероятный неудовлетворительный результат при условии, что количество признаков значительно больше числа образцов;
- отсутствие оценки, вероятности вычисляются с использованием дорогой пятикратной кросс-валидации;
- масштабирование данных.

3.3 Random forest – случайный лес

Алгоритм «*случайный лес*» использует много деревовидных классификаторов, делающих прогнозы, отсюда и название – лес. Алгоритм построен таким образом, что каждое дерево старается компенсировать недостатки других.

Преимущества:

- эффективность работы на больших данных;
- не требуется масштабирование данных;
- хорошо обрабатывает непрерывные и дискретные признаки;
- существование внутренней оценки значимости признаков;
- распараллеливание процесса.

Недостатки:

- склонность к переобучению, особенно на зашумленных данных.
- большой размер моделей.

3.4 Оценка эффективности модели

Поскольку будет проводиться сравнение различных классификаторов, необходимо ввести метрики, по которым будет вестись сравнение. Каждый отзыв обучающей выборки принадлежит одному из классов. Натренированный бинарный классификатор расставляет метки на тестовом множестве. По итогам тестовая выборка разбивается на 4 группы. Их размеры характеризуют работу классификатора на данной выборке.

- TP (true positive) – истинно положительные;
- TN (true negative) – истинно отрицательные;
- FP (false positive) – ложно положительные;
- FN (false negative) – ложно отрицательные.

Для оценки вводятся показатели точность (precision) и полнота (recall):

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

Для оценки классификатора требуется только одно число, характеризующее его работу. В качестве такой метрики можно взять -меру, которая позволяет найти баланс между точностью и полнотой и представляет собой их среднее гармоническое взвешенное.

$$F_\beta = \frac{(\beta^2 + 1)}{\beta^2} \frac{PR}{P + R}$$

Значение F лежит в промежутке [0, 1] и зависит от значения β, при значении коэффициента $\beta < 1$ предпочтение отдается полноте, а при $\beta > 1$ – точности. Недостатком такой метрики является то, что требуется фиксированное значение параметра. Существуют метрики лишенные данного недостатка, например, *AUC* (*area under ROC curve*) – площадь под кривой

ошибок. Кривая *ROC* (*receiver operating characteristic*) представляет собой зависимость доли истинно положительных (*TPR – true positive rate*) от доли ложно положительных результатов (*FPR – false positive rate*).

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{TN + FP}$$

ROC кривую изображают на графике, эта функция всегда возрастает, больший изгиб говорит о лучшей работе бинарного классификатора. Визуально достаточно сложно сравнить эффективность нескольких моделей, поэтому сравнивается численный показать – площадь под кривой ошибок – *AUC*. Значение *AUC* также лежит в диапазоне [0, 1]. Чем выше значение метрики, тем лучше эффективность модели. Ее и будем использовать для сравнения работы классификаторов.

Глава 4. Экспериментальная часть

4.1 Данные

Для экспериментального сравнения языковых моделей была выбрана задача определения тональности рецензий на русском языке пользователей сервиса *Kinopoisk*. Имеющиеся данные представляют собой 22379 отзывов, из которых 17838 имеют положительную оценку, 4541 – отрицательную. Также существуют нейтральные отзывы, которые не несут специфической эмоциональной окраски. Но было принято решение не включать такие данные, так как последующая оценка моделей не для бинарного классификатора, является весьма трудной задачей. Для загрузки рецензий была реализована программа на языке Python, выполняющая парсинг с сайта www.kinopoisk.ru [6].

В качестве рабочей машины использовался ноутбук Sony Vaio с процессором Intel Core i5 – 2450M CPU @ 2.50 GHz × 4. В качестве интерактивной веб-оболочки использовался *Jupyter Notebook*, язык программирования – *Python* (*ver. 2.7*). Для хранения данных была выбрана реляционная СУБД – *MySQL*, которая включает в себя 2 таблицы: `*film*`, `*review*`. Таблица `*film*` содержит поля `*id*`, `*title*`, `*review*` – `*id*`, `*id_film*`, `*tone*`, `*review*`.

Пример одного из отзывов:

```
  Картина российского режиссера Виктора Демента. Сильная психологическая драма о непростых отно
шениях инспектора рыбонадзора Трофима Русанова с окружающими его людьми и, в первую очередь,
с самим собой. Живущий по принципу «все по закону» и навязывая свою линию поведения обществу,
в силу неожиданного происшествия, вынужден полностью изменить свой взгляд на мир.  С самых перв
ых кадров зрителю сложно воспринять главного героя, как положительного. С одной стороны, челове
к доблестно, беспрекословно исполняет свой долг, но с другой — зачастую доводит некоторые ситуа
ции до абсурда, выходя за грани разумного, тем самым раздражая людей и вызывая у них лишь отвра
щение. «Чужой среди чужих»: ни с кем не общается, никого не слышит и не слушает, даже с женой,
за тридцать лет совместной жизни — ни разу не говорил по душам.  Стоит отметить выбор актера н
а главную роль. Алексей Гуськов феноменально передал характер и поведение главного героя. Зачас
тую, оставаясь один на один со зрителем, не произнося ни слова, он выдает настолько сильные вну
тренние монологи, настолько умело жонглирует эмоциями, выстраивая кардиограмму категорически ра
зных состояний, что едва ли найдется человек, который не прочувствует все это вместе с ним. О
н будто ведет зрителя за собой, вдоль темного холодного озера, которое из года в год тянет ег
о все глубже, в трясину своей слепой ненависти ко всему, в заснеженный, суровый лес реальност
и — заставляет участвовать в своих переживаниях. Плоды работы оператора картины, Андрея Найдено
ва, дополняют все это визуальным, объемным восприятием природы и пейзажей. Бескрайний лес и в т
о же время жесткие рамки. Казалось бы, простор и свобода, но на деле — лес душит, запирает геро
я в рамках своих же собственных предубеждений и ими же губит его до тех пор, пока ему не открое
тся новый, совершенно иной мир.  Конечно, предварительно не прочитав одноименную повесть Влади
мира Тендрякова, явившуюся отправной точкой для создания этой картины, сложно уловить некоторые
детали. Однако режиссер умело компенсирует это сильными эмоциональными сценами, сгущая краски
и даже утрируя развитие событий, а очень правильно и аккуратно подобранная музыка идеально асси
милируется с визуальным рядом и общим настроением.
```

В таком виде отзывы использовать нельзя, они содержат много мусора, для начала их нужно предварительно обработать. Будем использовать библиотеку *NLTK* (*Natural Language Toolkit*) – библиотека для обработки естественного языка. Когда решаем, как чистить текст, необходимо думать о решаемой задаче. Иногда имеет смысл удалить все знаки препинания. С другой стороны, в нашем случае, речь идет об анализе тональности текста и, возможно, «!!!» или «☹» могут нести эмоциональную окраску, тогда их следовало бы рассматривать как слова. Для простоты удалим все знаки препинания, приведем все слова к нижнему регистру, а также избавимся от всех чисел. Наконец нужно решить, как быть с часто встречающимися словами, которые не несут особого смысла (в, на, и, а, …), такие слова называют «стоп-словами». В зависимости от выбранной модели будет принято решение, удалять их или оставить в тексте. В *Python* существуют пакеты, которые содержат списки «стоп-слов» на различных языках, в том числе и на русском языке.

```
и в во не что он на я с со как а то все она так его но да ты к у же вы
за бы по только ее мне было вот от меня еще нет о из ему теперь когда
даже ну вдруг ли если уже или ни быть был него до вас нибудь опять уж
вам ведь там потом себя ничего ей может они тут где есть надо ней для
мы тебя их чем была сам чтоб без будто чего раз тоже себе под будет ж
тогда кто этот того потому этого какой совсем ним здесь этом один почт
и мой тем чтобы нее сейчас были куда зачем всех никогда можно при нако
нец два об другой хоть после над больше тот через эти нас про всего ни
х какая много разве три эту моя впрочем хорошо свою этой перед иногда
лучше чуть том нельзя такой им более всегда конечно всю между
```

Количество положительных и отрицательных отзывов сильно отличается, поэтому при обучении будет использоваться *перекрестная проверка* (*cross-validation*). Идея заключается в том, что все множество данных нужно разбить на k частей, при этом обучение проводится на $k-1$ части, а на оставшейся – тестирование. Таким образом, обучение модели проводится k раз.

4.2 Словарь тональности

Для русского языка в настоящий момент имеется не так много размеченных данных. Но все же такие словари существуют, для использования будем применять словарь, который взят с ресурса *linis-crowd.org*. Объем корпуса составляет более 7000 слов, где каждое представлено в инфинитиве и имеет определенную эмоциональную окраску. Градация тональности слов находится в пределах от -2 до 2, от крайне отрицательной оценки до крайне положительной.

Для реализации этого метода потребуется привести все слова из отзывов к начальной форме. Для этого в *Python* существует библиотека *Pymorphy2*. Далее, для каждого термина из словаря высчитывается его вес. Тональность текста определяется взвешенной суммой слов.

К сожалению, корпус, на котором был основан словарь, и лексика, используемая в рецензиях, очень сильно разнятся, поэтому высокой точности добиться не удалось. Точность метода взвешенной суммы слов по метрике AUC представлена на Рис. 11 [7].

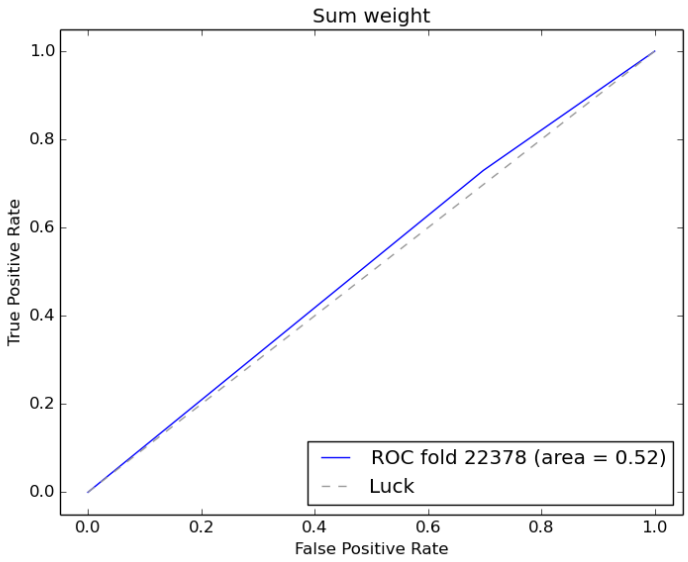

Рис. 11. Точность метода взвешенной суммы слов по метрике AUC.

4.3 LSTN-RNN

Для обучения рекуррентной нейронной сети LSTM данные необходимо представить в числовом виде, каждому документу соответствует свой вектор. Сделаем это следующим образом. Почистим все отзывы, «стоп-слова» удалять не будем. Затем заменим одинаковые слова на одинаковые числовые коды. Получился корпус примерно из 200000 слов, а каждый отзыв представлен в виде вектора. Теперь можно начать классификацию. В *Python* существует много библиотек, где можно реализовывать нейронные сети. В данной работе будет использоваться библиотеками *Theano* и *Keras*. *Keras* – модульная библиотека нейронных сетей, библиотека глубинного обучения для *Theano*. Реализация нейронной сети соответствует описанной ранее архитектуре.

Векторное представление рецензий было ограничено по длине, в итоге в каждом векторе осталось по 100 компонент. Обучение нейронной сети проходило в 5 эпох. Время обучения составило 3556.55 секунд. Достигнутая точность по метрике AUC составила 0.997 (Рис. 12).

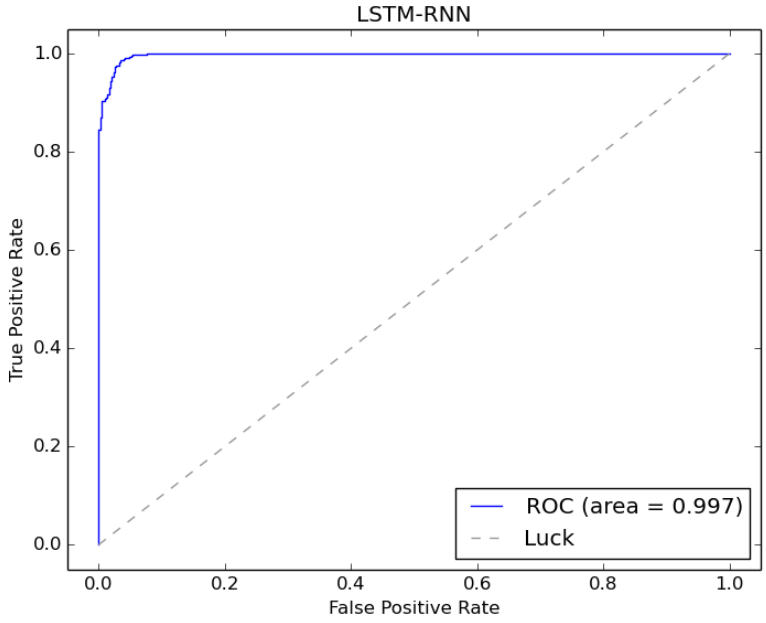

Рис. 12. Точность классификатора LSTM-RNN по метрике AUC.

4.4 «Мешок слов»

Для векторного представления документов воспользуемся моделью «*мешок слов*». Количество отзывов в выборке превышает 20000, поэтому заменять каждый документ вектором с учетом мощности словаря будет проблематично. Для решения этой проблемы уменьшим размер словаря, оставив только N наиболее часто встречающихся слов. Так как ситуация вынуждает выкинуть часть слов, перед составлением словаря необходимо удалить из отзывов все «стоп-слова». Мощность словаря возьмем $N = 5000$. В итоге получим набор документов, где каждый представлен вектором размерностью 5000.

4.5 Word2vec

Будем использовать реализацию *Word2Vec* в *Python* от *Gensim package*. Обе версии от *Google* и от *Gensim* полагаются на многопоточность – запуск нескольких процессов параллельно для экономии времени. Для того чтобы обучить модель в разумные сроки, необходимо установить *Cython*. *Word2vec* будет работать и без установки *Cython* на компьютер, но это займет несколько дней, вместо нескольких минут.

Перейдем к обучению. Как и в случае с моделью «*мешок слов*», требуется почистить данные, но в данном случае есть несколько отличий. Тренировать *Word2Vec* лучше без удаления «стоп-слов», так как алгоритм опирается на широкий контекст в предложении для получения высококачественных векторов слов. *Word2Vec* предполагает получить на вход отдельные предложения, представленные в виде списка слов. Иными словами формат ввода – список списков. Разделение текста на предложения является трудоемкой задачей. Существует много видов подводных камней в естественных языках. Предложения могут заканчиваться различными знаками препинания «?», «!», «.» в том числе и другими. По этой причине будем использовать *punkt* из *NLTK* для разделения текста на предложения.

При создании модели *Word2Vec* можно определить не только ее архитектуру, но и другие не менее важные параметры:

- архитектура: *Skip-gram* (по умолчанию) и *CBOW* (*Continuous Bag-of-Words*, *непрерывный* «*мешок слов*»);
- обучающий алгоритм: *иерархический softmax* (по умолчанию) или *негативное семплирование* (*negative sampling*);
- субдискретизация частых слов (*sample*). Документация *Google* рекомендует значения в пределах $[0.0001, 0.001]$;
- размерность векторов слов (*size*). Больше функций приводит к более продолжительному времени работы и часто, но не всегда, к

лучшему результату. Разумные значения могут быть от нескольких десятков до нескольких сотен;

- размер контекстного окна (*window*). Чем больше слов из контекста будет учитывать алгоритм для обучения, тем лучше, но до определенного момента;
- рабочие потоки (*workers*): количество параллельных процессов для запуска. Этот параметр зависит от компьютера, но в промежутке от 4 до 6 должно работать на большинстве систем;
- минимальное количество слов (*min_count*). Этот параметр позволяет ограничить размер корпуса. Слова, которые встретились мало раз во всех документах, будут игнорироваться. Разумные значения могут быть в пределах от 10 до 100.

Выбрать параметры нелегко, но как только выбор сделан, создание модели *Word2Vec* будет весьма простым. Для обучения были выбраны следующие параметры:

Skip-gram, softmax, workers = 4, size = 300, min_count = 40, window = 10, sample = 1e-3. Обучение *Word2Vec* составило порядка 88.8 секунд, в корпусе содержится 18610 терминов, каждое слово представлено в виде вектора в 300-мерном пространстве. Размер сохраненной модели – 46.3 МБ.

Приведем несколько примеров семантического калькулятора.

Определим запрос, в котором выполняются следующие векторные операции:

$$vec(\text{"король"}) - vec(\text{"мужчина"}) + vec(\text{"женщина"}),$$

предположительным результатом будет $vec(\text{"королева"})$. Посмотрим, какие результаты покажет модель.

Запрос:

```
print(model.most_similar(positive = [u'король', u'женщина'],
                         negative = [u'мужчина']))
```

Результат:

```
девушка 0.589871168137
сестра 0.572908103466
королева 0.566203653812
этакая 0.522121191025
роза 0.516714155674
фрейя 0.51347386837
семейка 0.513338267803
подружка 0.511786043644
подруга 0.508782148361
комната 0.498883664608
```

Также из любого набора слов можно легко убирать лишнее простым сравнением векторов. Удаляются те слова, векторы которых наиболее удалены от среднего. Так как модель была построена на рецензиях к фильмам, составим запрос, близкий к этой тематике.

Запрос:

```
print(model.doesnt_match(u'бенедикт фриман дикаприо мориарти'.split()))
```

Результат:

```
дикаприо
```

Word2vec может находить близкие слова к заданному.

Запрос:

```
print(model.most_similar(u"шерлок"))
```

Результат:

```
холмс 0.721418559551
камбербэтч 0.578717529774
бенедикт 0.570861756802
билл 0.51461994648
леонид 0.50624537468
джобс 0.505993783474
багира 0.498874127865
сыгранный 0.493009030819
сергей 0.492778480053
данила 0.491948962212
```

Запрос:

```
print(model.most_similar(u"дикаприо"))
```

Результат:

```
лео 0.729911625385
ди 0.696039974689
леонардо 0.631466090679
каприо 0.626152932644
тома 0.499240577221
хью 0.475234895945
харди 0.416354417801
оскара 0.412327736616
любецки 0.4066170156
отыграл 0.406066089869
```

Теперь перейдем к следующей части, к определению тональности рецензий. Существует несколько способов реализации. После использования модели *Word2Vec* каждое слово представлено в виде вектора. Необходимо их объединить, чтобы можно было классифицировать документы по тональности.

Векторное усреднение

Одной из проблем является то, что все отзывы имеют переменную длину. Необходимо найти способ, чтобы взять отдельные векторы слов и превратить их в набор особенностей, который имеет одинаковую длину для всех рецензий. Поскольку каждое слово это вектор в *N*-мерном пространстве, мы можем использовать векторные операции, чтобы объединить слова в каждом обзоре. Один из способов – это просто усреднить векторы слов в

данном отзыве. Для этой цели мы удалим «стоп-слова», которые просто добавляют шум. После того, как мы усреднили векторы, вернемся к задаче классификации для разметки данных.

Кластеризация

Word2vec создает кластеры семантически связанных слов, поэтому еще один возможный подход заключается в использовании сходств слов внутри кластера. Группировка векторов таким способом известна как «*векторное квантование*» («*vector quantization*»). Для этого сначала нужно найти центры кластеров – слова, можно сделать это при помощи алгоритма кластеризации, например, *k-means*. В *k-means* только один параметр, который нужно установить, это k – количество кластеров. Как решить, сколько кластеров создавать? Методом проб и ошибок! Предположим, что небольшие кластеры дают лучший результат, нежели большие. Будем использовать библиотеку *scikit-learn* для выполнения кластеризации *k-means*. Алгоритм с большим k может быть очень медленным, поэтому уставим таймер, для замера времени на выполнение кластеризации. Также проверим предположение о величине кластеров, для сравнения возьмем среднее количество слов в кластере 50 и 10.

Возьмем сначала за среднее значение 50, тогда количество кластеров, то есть k будет определяться следующим образом: $k = \frac{\text{мощность словаря}}{50}$. Запустим алгоритм кластеризации, при данном параметре время разбиения составило 521.32 секунд, $k = 372$.

Это немного абстрактно, но стоит взглянуть на содержимое кластеров.

```
Cluster 0
здорово достойно вписывается гармонично максимально органично замечательно превосходно шикарно
великолепно грамотно мастерски убедительно неплохо отлично прекрасно хорошо задачей удачно безу
пречно потрясающе невероятно блестяще справляется идеально ярко подобран

Cluster 1
закончилось перевод словами бред серьезно типа простите спойлеры давайте короче получится ох «ч
то мягко богу «почему бла «как «кто вроде скажем вкратце неправильно !» коротко возьми ха навер
но «и но… немножко названия дай и… черт прям «если ах боже «вы правильно точнее … «вот кстати л
адно собственно «зачем блин опять посмотрим сюда «он насчет извините «мы ниже «это ка – слава ч
ёрт

Cluster 2
средств концепции подачи первоисточника режиссуры съемок сопровождения анимации размышлений вос
приятия реализации эффекта режиссерской исполнения использования группы шедевра музыкального ус
пеха эммануэля визуального авторов оригинальности публики визуальных создания съемочной актёрск
ого оригинала используя сценария команды дебюта материала саундтрека общей опыта режиссерского
постановки художественной содержания музыки повествования композитора таланта съёмочной «выживш
его» стиля съёмок процесса продукта творения монтажа продюсеров автора специально мастерство ма
стерства бюджета музыкальной проекта
```

Аналогично за среднее значение возьмем 10. При данном параметре время разбиения на кластеры составило 7550.72 секунд, $k = 1861$. Содержимое кластеров выглядит следующим образом.

```
Cluster 0
желанием стремление

Cluster 1
дорог вспомнит читает подозревает узнав понимая рассказал признается пишет прав зная кричит ист
ину вспоминает заявляет думая осознал осознает мечтал помнит заявил узнаёт убивал забывает заду
мывается курсе искал бросил занимается рассказывал знающий придет память

Cluster 2
убили вскоре маленькую отправляет любимую заодно меч похоронить дочку рукой тайну задание остав
ив оттуда навсегда родную соглашается надежде родину тюрьму встречу принял жертву маму дорогу у
ехать квартиру чистую чудом школу письмо деревню желая подругу световой решается девочку обратн
о машину миссию просит джедая сестру

Cluster 3
ожиданий спойлеры заключении трейлеры мною причин пойду постер заранее прочитав итак дважды отз
ыва зря мою итоги критику б сегодня

Cluster 4
рисует представляя ломает вводят активно оборачивается целое выводит вводит погрузить шокирует
удерживает искусно переносят психику плотно вечную превратившись пучину выбирая проникает прони
кнуть погружаясь убеждает ткань погружая насквозь тоску выставляя возвращает открыв прочно водо
ворот всецело перерастает заманивает покоряет рождает превращаясь гущу попадая надолго среду мо
ментально рассматривает наводит страхе погружения разрушает рушит окружает внутрь сопровождает
окунает уносит воображение воображении играя подобно суровую достигает чёрно подчёркивает охват
ывает поглощает пронизывает сохраняет преображается ловко постоянном погружают мгновенно давит
переносит описывает дышит иллюзию наизнанку наполняет становясь пред демонстрируя выстраивает о
кутывает погружается превратив искушенного настраивает
```

Результаты весьма интересные, но теперь есть кластер или центроида для каждого слова. Можно определить функцию для преобразования отзыва в мешок центроид. Это похоже на «*мешок слов*», но здесь используются семантически связанные кластеры, а не отдельные слова. После получаем вектор для каждого кластера с множеством особенностей, равных количеству кластеров. Теперь можно переходить к классификации.

4.6 Doc2Vec

Для обучения модели *Doc2Vec* рецензии были почищены: удалены знаки препинания, числа, все буквы приведены к нижнему регистру. «Стоп слова» удалять не будем, так как модель *Doc2Vec*, как и *Word2Vec*, ориентируется на контекст. На этом предварительная обработка данных не заканчивается. Модель *Doc2Vec* принимает на вход объекты класса *LabeledSentence*. Каждый документ преобразуем в объекты нужного класса, можно сказать, что каждый документ представлен в виде списка слов и уникальной метки. В нашем случае метки будут делить данные на 4 класса: тренировочное и тестовое, а каждый из них подразделяется по тональности документа. Авторы метода рекомендуют использовать объединенные векторные представления обеих архитектур, ссылаясь на получение векторов более высокого качества. Так и поступим. Создадим 2 модели для обучения: *DM* и *DBOW*. При создании можно определить параметры моделей, которые будут для обеих одинаковыми. Размерность получаемых векторов определим в 400-мерном пространстве, размер контекстного окна возьмем в 10 слов, субдискретизация частых слов (*sample* = 1e-3), количество рабочих потоков – 4. Обучать модели будем в несколько эпох. В рекомендациях указывается, что *Doc2Vec* лучше обучать на данных, поступающих в случайном порядке, либо на каждой итерации изменять скорость обучения. В данной работе используется понижение скорости обучения. Обучение обеих моделей происходило одновременно и проходило в 10 эпох, по времени это заняло 8744.74 секунд.

Интересно посмотреть, какие результаты выдает модель *Doc2Vec* в ответ на запросы, которые были продемонстрированы для *Word2Vec*.

Запрос:

```
print(model_dm.most_similar(positive = [u'король', u'женщина'],
                            negative = [u'мужчина']))
```

Результат *D2V-DM*:

```
ошалевший 0.240217551589
арто 0.239317134023
девушка 0.236558556557
женщина… 0.230106770992
«пережевывается» 0.22412751615
комедия 0.217799559236
развратность 0.217584744096
собственников» 0.21706160903
сломлен 0.214360266924
раздался 0.208961427212
```

Результат *D2V-DBOW*:

```
юга! 0.211820438504
исаака 0.208121284842
«опустел 0.208103552461
декоративный 0.2058185637
вещающем 0.197355031967
вырывая 0.197325766087
задания 0.196793407202
сознававшего 0.194295153022
шарьера 0.192786663771
улетающей 0.190118581057
```

Запрос:

```
print(model_dbow.most_similar(u'шерлок'))
```

Результат *D2V-DM*:

```
вывернуться 0.268470197916
ежеминутная 0.265925854445
«семеро 0.249124661088
пацаном 0.247482821345
городе! 0.246896952391
страданий… 0.237156406045
дознавание 0.237104013562
позволить… 0.236290618777
своевременности 0.231805294752
перевязанными 0.230436354876
```

Результат *D2V-DBOW*:

```
грива 0.213016808033
рубена 0.208374232054
покурить 0.203946441412
крутились 0.201822146773
реализованные 0.198325872421
протянули 0.196677193046
воспылал 0.196057111025
встречаетесь 0.195024058223
вы 0.193700700998
патриотическую 0.193543195724
```

Запрос:

```
print(model_dbow.doesnt_match(u'бенедикт фриман мориарти дикаприо'.split()))
```

Результат *D2V-DM* и *D2V-DBOW*:

```
дикаприо
```

Получились весьма необычные результаты, возможно, это связано с небольшим количеством эпох при обучении. Теперь перейдем к задаче классификации.

4.1 Обучение классификатора

Рассмотрим несколько классификаторов, используя векторное представление документов, составленное при помощи модели «*мешок слов*», сравним их, применив *ROC-анализ*. Сейчас есть набор документов и тональность для каждого. При кросс-валидации за k возьмем $k = 10$. Теперь можно начать обучение с учителем.

SVM (support vector machine) – метод опорных векторов

Будем использовать библиотеку в *Python – scikit-learn*. Библиотека включает в себя различные реализации классификации, регрессии и кластеризации алгоритмов. Также имеется предварительная обработка данных, поэтому воспользуемся встроенной функцией *scale()* для масштабирования векторов. В качестве входных данных для обучения классификатора будем менять следующие параметры: ядро *kernel*, максимальное количество итераций *max_iter*.

Random Forest – случайный лес

Этот классификатор также имеется в библиотеке *scikit-learn*. В качестве параметров можем устанавливать количество деревьев. По умолчанию стоит 100. Большее количество деревьев могут работать лучше, а могут и хуже, но, безусловно, это займет больше времени для запуска алгоритма.

Для обучения классификаторов устанавливались различные параметры, лучшие результаты представлены на Рис. 13. Параметры для классификаторов: *SVM* (*kernel = linear*, *max_iter = 100*), *Random Forest* (*n_estimators = 100*). Результаты экспериментов приведены в *приложении 1*.

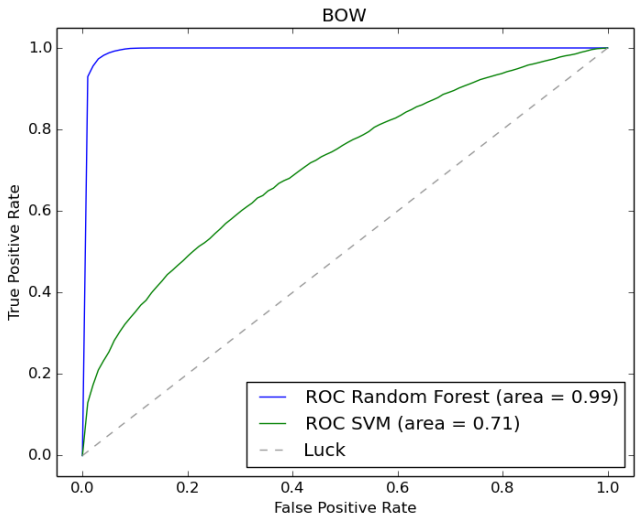

Рис. 13. Точность классификаторов по метрике AUC для модели BOW (N = 5000). Классификаторы SVM (kernel = linear, max_iter = 100), Random Forest (n_estimators = 100).

Для модели *Word2Vec* и усредненных векторов документов были достигнуты следующие результаты (Рис. 14). Параметры для классификаторов: *SVM* (*kernel = rbf, max_iter = 500*), *Random Forest* (*n_estimators = 100*). Результаты экспериментов приведены в *приложении 2*.

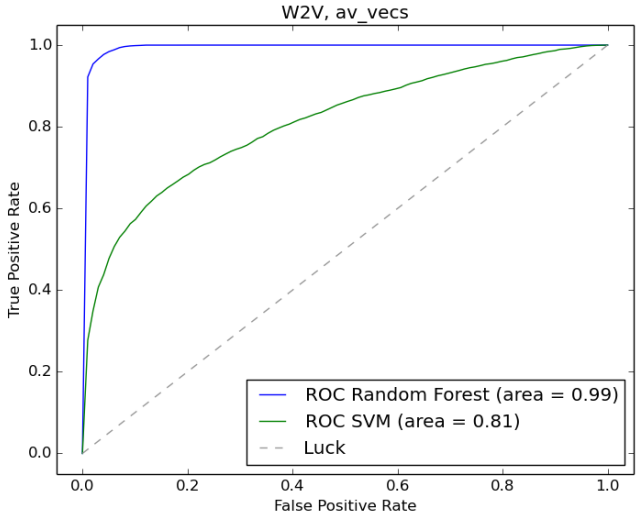

Рис. 14.Точность классификаторов по метрике AUC для модели W2V и усредненных векторов слов. Классификаторы SVM (kernel = rbf, max_iter = 500), Random Forest (n_estimators = 100).

Для модели *Word2Vec* и центроид кластеров при $k = 372$ были достигнуты следующие результаты (Рис. 15). Параметры для

классификаторов: *SVM* (*kernel = rbf*, *max_iter = 500*), *Random Forest* (*n_estimators = 100*). Результаты экспериментов приведены в *приложении 3*.

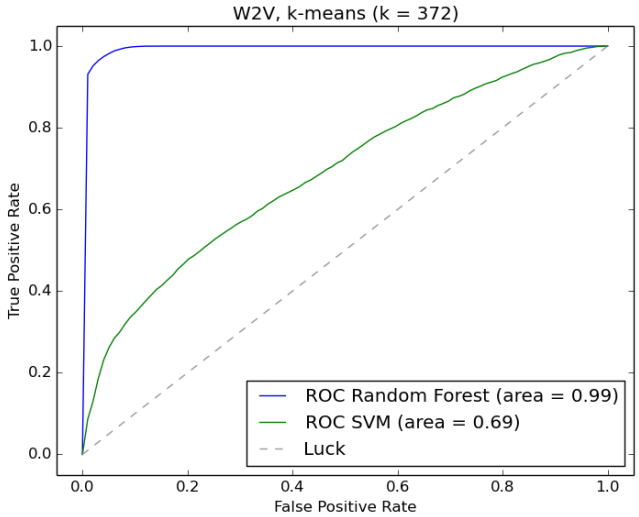

Рис. 15.Точность классификаторов по метрике AUC для модели W2V и центроид кластеров при k = 372. Классификаторы SVM (kernel = rbf, max_iter = 500), Random Forest (n_estimators = 100).

Для модели *Word2Vec* и центроид кластеров *k-means* ($k = 1861$) были достигнуты следующие результаты (Рис. 16). Параметры для классификаторов: *SVM* (*kernel = rbf*, *max_iter = 500*), *Random Forest* (*n_estimators = 100*). Результаты экспериментов приведены в *приложении 4*.

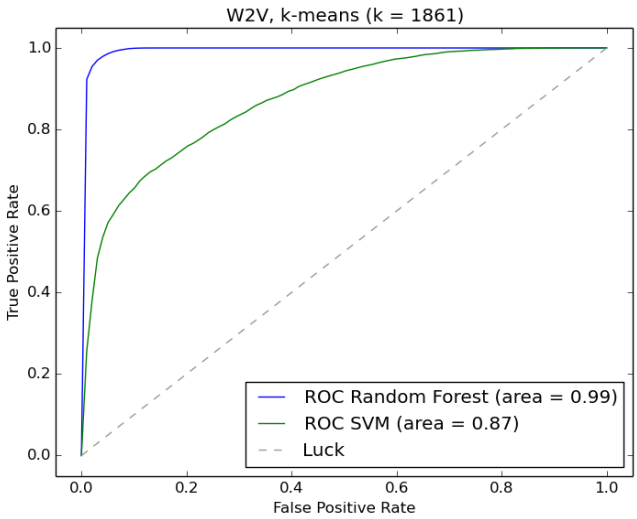

Рис. 16.Точность классификаторов по метрике AUC для модели W2V и центроид кластеров при k = 1861. Классификаторы SVM (kernel = rbf, max_iter = 500), Random Forest (n_estimators = 100).

Классификатор «случайный лес» зарекомендовал себя, как наиболее эффективный, для решения поставленной задачи – определение тональности текста. Объединенные векторы двух моделей *D2V-DM* и *D2V-DBOW* были поданы на вход для обучения классификатора *Random Forest*, количество деревьев = 100. Обучение составило 79.8 секунд, но в данном случае перекрестная проверка не выполнялась. На Рис. 17 представлена точность метода по метрике *AUC*.

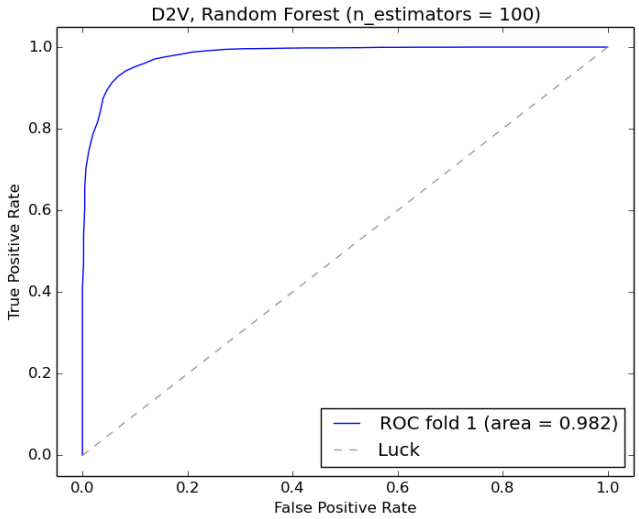

Рис. 17. Точность классификатора Random Forest (n_estimators = 100) для объединенных векторов моделей D2V-DM и D2V-DBOW по метрике AUC.

Представим наглядно точности различных классификаторов, примененных для разных моделей, по метрике *AUC* (на Рис. 18 приведены только лучшие результаты).

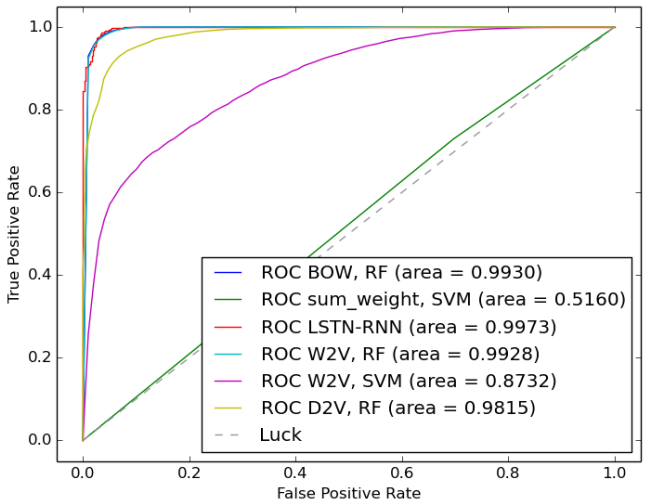

Рис. 18. Точность различных классификаторов, примененных для разных моделей, по метрике AUC.

Выводы

На основании результатов решения задачи определения тональности рецензий пользователей сервиса *Kinopisk* можно сделать следующие выводы:

- метод определения тональности, основанный на словаре, неуниверсален: мало размеченных данных, слова в разных контекстах могут нести разную тональность;
- определение тональности с использованием рекуррентной нейронной сети *LSTM* показало наилучший результат, но неясно, насколько хорошо будет работать данная структура на незнакомых данных;
- модель «*мешок слов*» работает хорошо, но при большем количестве данных увеличится размер корпуса, следовательно, и размерность векторов, что приведет к высокой вычислительной сложности;
- модели *Word2Vec* и *Doc2Vec* имеют несомненное преимущество перед другими моделями, в частности, существенное понижение размерности признаков.

Заключение

В данной работе проводилось исследование методов глубинного изучения и их применение к задачам обработки текстов. Был произведен обзор современных методов машинного обучения, основанных на применении многослойных нейронных сетей. Для сравнения были получены результаты, основанные на традиционном подходе, для определения тональности текстов – метод, основанный на словаре. Данный подход является неуниверсальным, так как требуется очень много размеченных данных, также тональность слов может меняться в зависимости от контекста. Другие рассмотренные модели используют векторное представление слов или документов. Наиболее популярная и эффективная модель «мешок слов» показала очень хорошие результаты, но расширение корпуса приведет к увеличению вычислительной сложности. Если этого не делать, то при поступлении новых данных, будет падать точность алгоритма. Безусловно, работа нейросетевых моделей, показала очень хороший результат. К более традиционным моделям относится рекуррентная нейронная сеть *LSTM*, но неясно, насколько хорошо будет работать данная структура на незнакомых данных. Другой моделью является *Word2Vec*, которой в основном и была посвящена работа, имеющая преимущества перед остальными. Модель *Word2Vec* имеет на выходе связанные синтаксически и семантически векторные представления слов. Для сравнения различных методов определения тональности текстов в работе использовались рецензии к фильмам пользователей сервиса *Kinopoisk*. Полученные результаты, представленные на Рис. 18, практически идентичны для «*мешка слов*», *Word2Vec* и *LSTM-RNN*, но преимущество отдается *Word2Vec*, так как данная модель существенно понижает размерность вектора признаков, а также уменьшает вычислительные затраты при обучении. Также существует более общая модель для векторного представления слов – *Doc2Vec*, которая не только словам сопоставляет векторы, но и документам, поскольку достаточно

часто для применения различных методов машинного обучения требуется векторное представление для данных переменной длины.

Можно сделать основной вывод данной работы – нейросетевые модели являются передовыми в области обработки текстов на естественном языке.

Список литературы

1. Christopher Olah. Understanding LSTM Networks. http://colah.github.io
2. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. // In Proceedings of Workshop at International Conference on Learning Representations (ICLP) – 2013, http://arxiv.org/abs/1301.3781
3. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. Distributed Representations of Word and Phrases and their Compositionaly. // In Proceedings of Workshop at The Twenty-seventh Annual Conference on Neural Information Processing Systems (NIPS) – 2013, http://arxiv.org/abs/1310.4546
4. Tomas Mikolov, Quoc Le. Distributed Representations of Sentences and Documents. // In Proceedings of Workshop at The 31st International Conference on Machine Learning (ICML) – 2014, http://jmlr.org/proceedings/papers/v32/le14.pdf
5. Simon Haykin. Neural Networks a Comprehensive Foundation. Hamilton, Ontario, Canada, Pearson Education, Inc., 1999.
6. http://www.kinopoisk.ru
7. http://linis-crowd.org

Приложение

Приложение 1

Приведены результаты экспериментов для векторных представлений, полученных в результате работы модели «*мешок слов*» с ограничением размера корпуса в 5000 слов (Таблица 1).

Таблица 1. Результаты работы классификаторов на основе модели BOW (N = 5000).

algorithm	param_1	param_2	time	av_auc
SVM	kernel = linear	max_iter = 10	148 sec.	0.57
		max_iter = 50	738.6 sec.	0.62
		max_iter = 100	1541.83 sec.	0.71
	kernel = poly	max_iter = 10	136.2 sec.	0.52
		max_iter = 50	726.56 sec.	0.6
		max_iter = 100	1548 sec.	0.64
	kernel = rbf	max_iter = 10	158.2 sec.	0.49
		max_iter = 50	786.97 sec.	0.52
		max_iter = 100	1592 sec.	0.57
Random Forest	n_estimators = 100	-	739 sec.	0.99

Графическая интерпретация полученных результатов представлена на графиках ниже. Для классификатора *SVM* на Рис. 19, Рис. 20, Рис. 21 показана точность по метрике *AUC* для различных ядер и различного количества итераций при обучении классификатора. На Рис. 22 сравниваются лучшие показатели по количеству итераций для разных ядер. На Рис. 23 изображены результаты перекрестной проверки при обучении классификатора *Random Forest*.

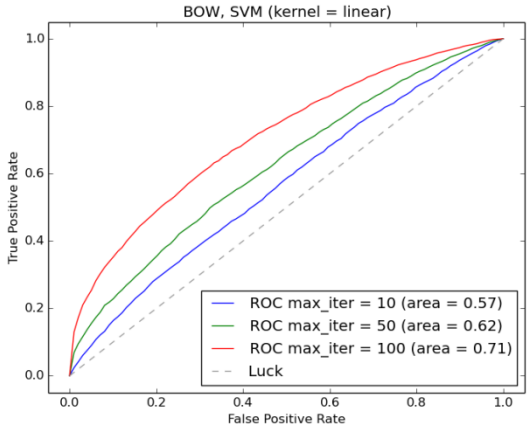

Рис. 19. Точность классификатора SVM (kernel = linear) по метрике AUC для модели BOW.

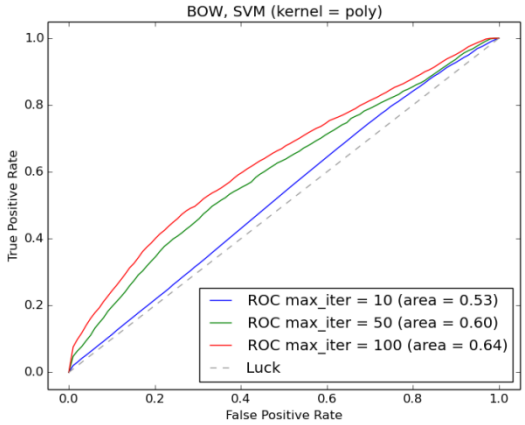

Рис. 20. Точность классификатора SVM (kernel = poly) по метрике AUC для модели BOW.

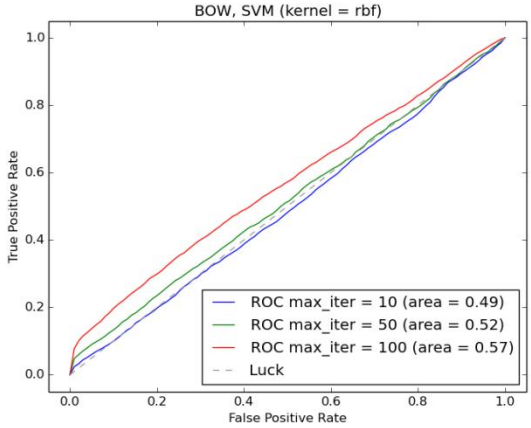

Рис. 21. Точность классификатора SVM (kernel = rbf) итераций по метрике AUC для модели BOW.

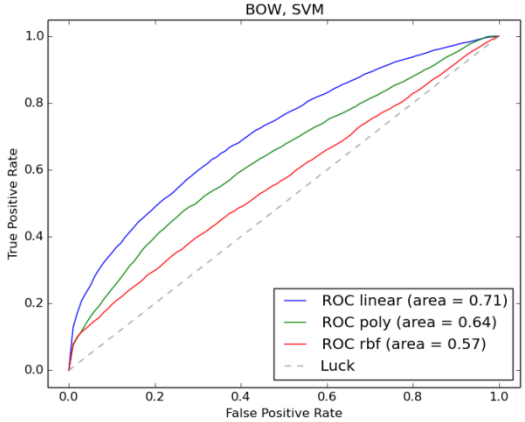

Рис. 22. Точность классификатора SVM для различных значений параметра kernel по метрике AUC для модели BOW.

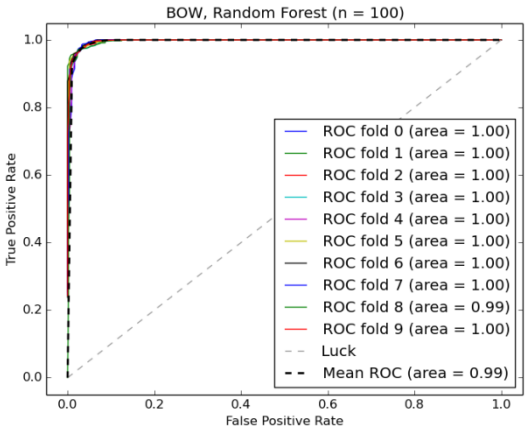

Рис. 23. Точность классификатора Random Forest (n_estimators = 100) для различных вариантов тестового и обучающего множества по метрике AUC для модели BOW.

Приложение 2

Приведены результаты экспериментов для векторных представлений, полученных в результате работы усредненных векторов слов, полученных при помощи модели *Word2Vec* (Таблица 2).

Таблица 2. Результаты работы классификаторов на основе модели W2V и усредненных векторов слов.

algorithm	param_1	param_2	Time	av_auc
SVM	kernel = linear	max_iter = 10	16.24 sec.	0.54
		max_iter = 50	49.47 sec.	0.61
		max_iter = 100	90.53 sec.	0.64
		max_iter = 500	334.18 sec.	0.65
	kernel = poly	max_iter = 10	11.14 sec.	0.55
		max_iter = 50	48.81 sec.	0.53
		max_iter = 100	95.21 sec.	0.55
		max_iter = 500	496.13 sec.	0.6
	kernel = rbf	max_iter = 10	12.15 sec.	0.58
		max_iter = 50	53.33 sec.	0.6
		max_iter = 100	105.11 sec.	0.65
		max_iter = 500	567.29 sec.	0.81
Random Forest	n_estimators = 100	-	262.5 sec.	0.99

Графическая интерпретация полученных результатов представлена на графиках ниже. Для классификатора *SVM* на Рис. 24, Рис. 25, Рис. 26 показана точность по метрике *AUC* для различных ядер и различного количества итераций при обучении классификатора. На Рис. 27 сравниваются лучшие показатели по количеству итераций для разных ядер. На Рис. 28 изображены результаты перекрестной проверки при обучении классификатора *Random Forest*.

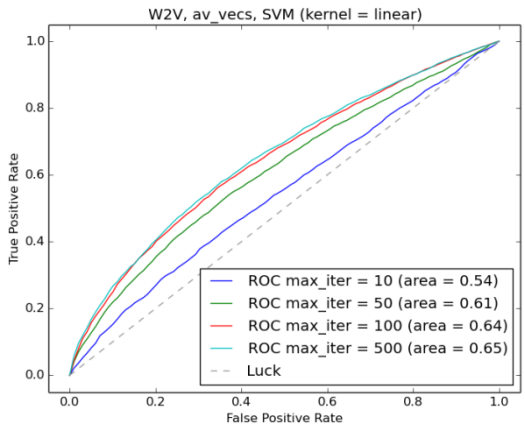

Рис. 24. Точность классификатора SVM (kernel = linear) по метрике AUC для модели W2V и усредненных векторов слов.

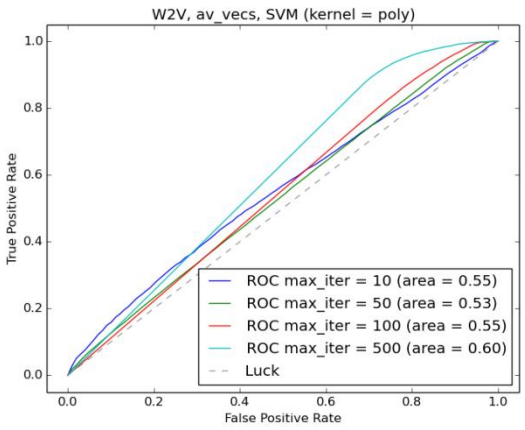

Рис. 25. Точность классификатора SVM (kernel = poly) по метрике AUC для модели W2V и усредненных векторов слов.

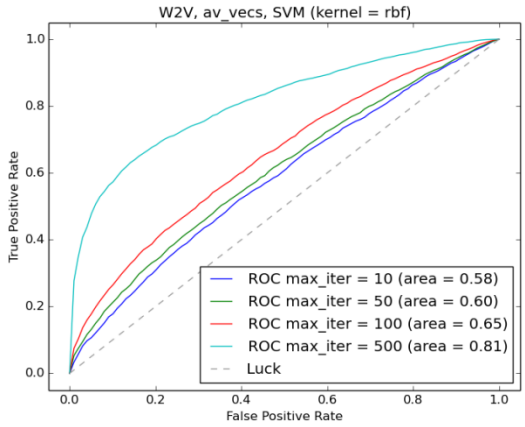

Рис. 26. Точность классификатора SVM (kernel = rbf) итераций по метрике AUC для модели W2V и усредненных векторов слов.

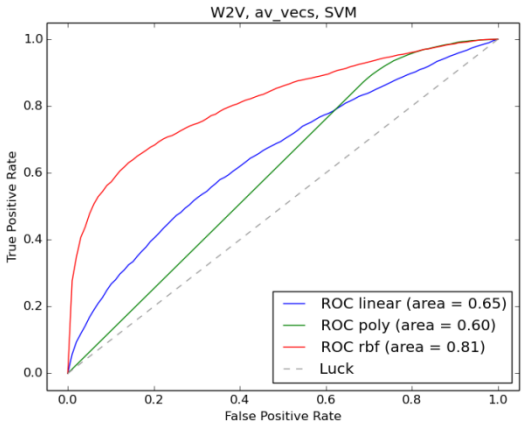

Рис. 27. Точность классификатора SVM для различных значений параметра kernel по метрике AUC для модели W2V и усредненных векторов слов.

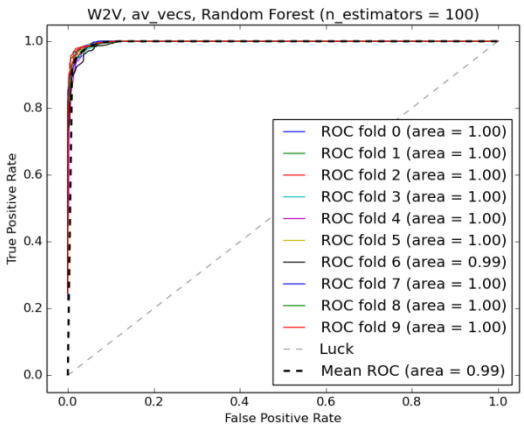

Рис. 28. Точность классификатора Random Forest (n_estimators = 100) для различных вариантов тестового и обучающего множества по метрике AUC для модели W2V и усредненных векторов слов.

Приложение 3

Приведены результаты экспериментов для векторных представлений, полученных в результате работы создания кластеров векторов при помощи алгоритма *k-means* ($k = 372$), полученных при помощи модели *Word2Vec*(Таблица 3).

Таблица 3. Результаты работы классификаторов на основе модели W2V и кластеризации k-means (k = 372).

algorithm	param_1	param_2	Time	av_auc
SVM	kernel = linear	max_iter = 10	13.9 sec.	0.55
		max_iter = 50	60.19 sec.	0.6
		max_iter = 100	118.31 sec.	0.61
		max_iter = 500	530.74 sec.	0.59
	kernel = poly	max_iter = 10	13.52 sec.	0.52
		max_iter = 50	60.89 sec.	0.54
		max_iter = 100	124.82 sec.	0.56
		max_iter = 500	662.72 sec.	0.62
	kernel = rbf	max_iter = 10	14.84 sec.	0.52
		max_iter = 50	65.28 sec.	0.54
		max_iter = 100	134.85 sec.	0.56
		max_iter = 500	956.33 sec.	0.69
Random Forest	n_estimators = 100	-	67.4 sec.	0.94

Графическая интерпретация полученных результатов представлена на графиках ниже. Для классификатора *SVM* на Рис. 29, Рис. 30, Рис. 31 показана точность по метрике *AUC* для различных ядер и различного количества итераций при обучении классификатора. На Рис. 32 изображены результаты перекрестной проверки при обучении классификатора *Random Forest*.

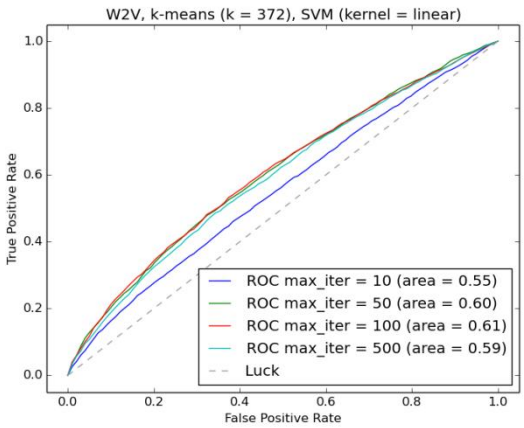

Рис. 29. Точность классификатора SVM (kernel = linear) по метрике AUC для модели W2V и кластеризации k-means (k = 372).

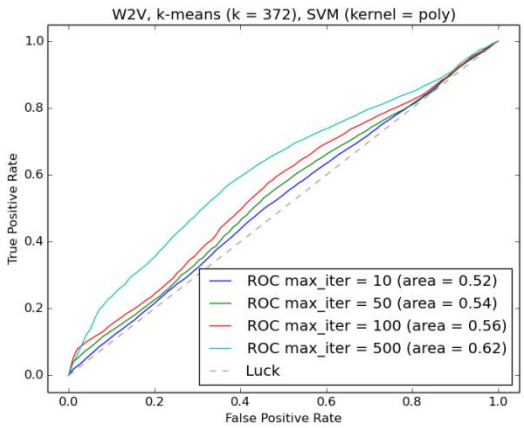

Рис. 30. Точность классификатора SVM (kernel = poly) по метрике AUC для модели W2V и кластеризации k-means (k = 372).

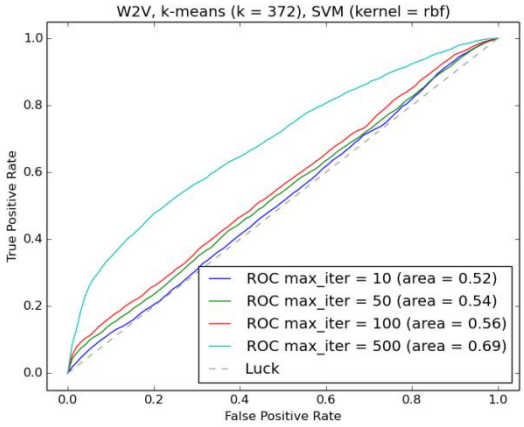

Рис. 31. Точность классификатора SVM (kernel = rbf) итераций по метрике AUC для модели W2V и кластеризации k-means (k = 372).

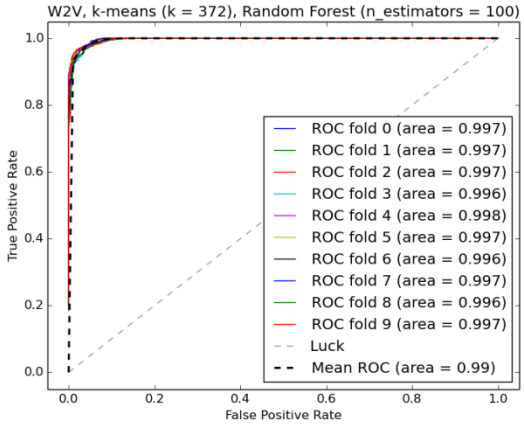

Рис. 32. Точность классификатора Random Forest (n_estimators = 100) для различных вариантов тестового и обучающего множества по метрике AUC для модели W2V и кластеризации k-means (k = 372).

Приложение 4

Приведены результаты экспериментов для векторных представлений, полученных в результате работы создания кластеров векторов при помощи алгоритма *k-means* ($k = 1861$), полученных при помощи модели *Word2Vec* (Таблица 4).

Таблица 4. Результаты работы классификаторов на основе модели W2V и кластеризации k-means (k = 1861).

algorithm	param_1	param_2	time	av_auc
SVM	kernel = linear	max_iter = 10	62.65 sec.	0.57
		max_iter = 50	217.25 sec.	0.66
		max_iter = 100	567.74 sec.	0.72
		max_iter = 500	2662.14 sec.	0.85
	kernel = poly	max_iter = 10	53.39 sec.	0.51
		max_iter = 50	268.55 sec.	0.57
		max_iter = 100	562.35 sec.	0.63
		max_iter = 500	2996.76 sec.	0.76
	kernel = rbf	max_iter = 10	62.69 sec.	0.5
		max_iter = 50	288.55 sec.	0.54
		max_iter = 100	603 sec.	0.58
		max_iter = 500	3090.23 sec.	0.87
Random Forest	n_estimators = 100	-	217.38 sec.	0.99

Графическая интерпретация полученных результатов представлена на графиках ниже. Для классификатора *SVM* на Рис. 33, Рис. 34, Рис. 35 показана точность по метрике *AUC* для различных ядер и различного количества итераций при обучении классификатора. На Рис. 36 сравниваются лучшие показатели по количеству итераций для разных ядер. На Рис. 37 изображены результаты перекрестной проверки при обучении классификатора *Random Forest*.

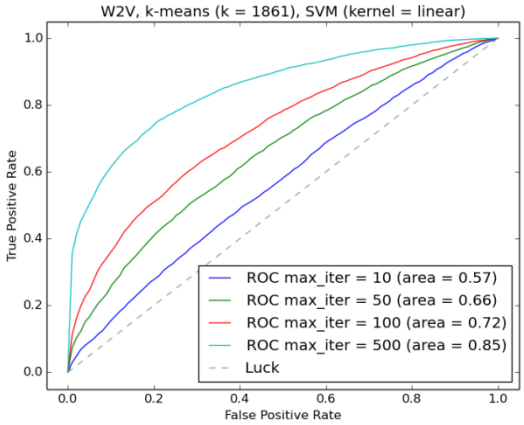

Рис. 33. Точность классификатора SVM (kernel = linear) по метрике AUC для модели W2V и кластеризации k-means (k = 1861).

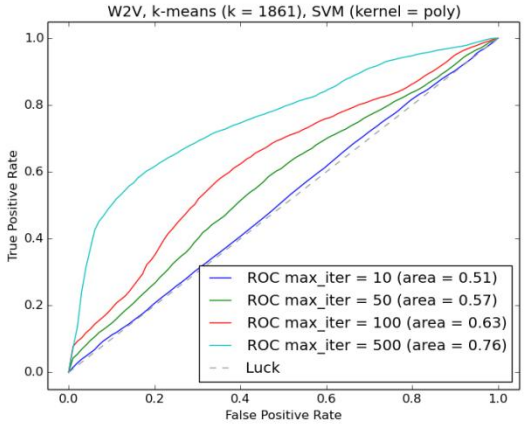

Рис. 34. Точность классификатора SVM (kernel = poly) по метрике AUC для модели W2V и кластеризации k-means (k = 1861).

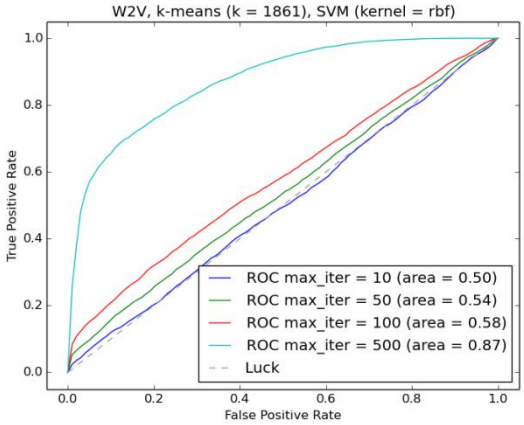

Рис. 35. Точность классификатора SVM (kernel = rbf) итераций по метрике AUC для модели W2V и кластеризации k-means (k = 1861).

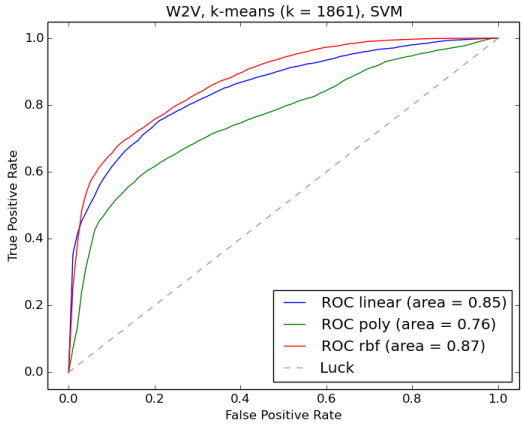

Рис. 36. Точность классификатора SVM для различных значений параметра kernel по метрике AUC для модели W2V и кластеризации k-means (k = 1861).

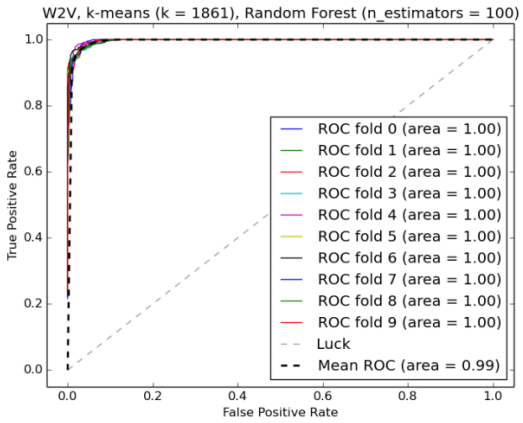

Рис. 37. Точность классификатора Random Forest (n_estimators = 100) для различных вариантов тестового и обучающего множества по метрике AUC для модели W2V и кластеризации k-means (k = 1861).